《数学机械化丛书》获国家基础研究发展计划项目“数学机械化方法及其在信息技术中的应用”与“数学机械化应用推广专用经费”资助

数学机械化丛书 9

交互式马尔可夫链

——并发系统的设计、验证与评价

吴尽昭　王永祥　覃广平　著

科 学 出 版 社

北 京

内 容 简 介

通过正交结合经典的进程代数和连续时间马尔可夫链模型，交互式马尔可夫链(IMC)提供了完美的可组合化的并发系统设计和分析框架. 本书主要介绍了IMC的理论及其在并发系统层次化设计以及功能验证与性能评价方面的应用，主要内容包括IMC上的分支时间等价和前序关系、IMC的动作细化理论以及基于IMC的模型检验方法.

本书可以供研究生、教师和科研人员作为了解并发系统理论研究及与数学机械化基本思想与方法相结合的形式在设计与分析技术最新成果的参考书.

图书在版编目(CIP)数据

交互式马尔可夫链：并发系统的设计、验证与评价/吴尽昭，王永祥，覃广平著. —北京：科学出版社，2007

(数学机械化丛书；9/吴文俊主编)

ISBN 978-7-03-018866-3

I.交… II. ①吴… ②王… ③覃 III. 马尔可夫链-研究 IV. 0211.62

中国版本图书馆 CIP 数据核字(2007)第 055039 号

责任编辑：鄢德平 赵彦超/责任校对：朱光光

责任印制：赵德静/封面设计：陈 敬

科 学 出 版 社 出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

中国科学院印刷厂 印刷

科学出版社编务公司排版制作

科学出版社发行 各地新华书店经销

*

2007 年 6 月第 一 版 开本：B5 (720×1000)

2007 年 6 月第一次印刷 印张：7 3/4

印数：1—3 000 字数：134 000

定价：28.00 元

(如有印装质量问题，我社负责调换〈科印〉)

《数学机械化丛书》前言[①]

十六七世纪以来, 人类历史上经历了一场史无前例的技术革命, 出现了各种类型的机器, 取代各种形式的体力劳动, 使人类进入一个新时代. 几百年后的今天, 电子计算机已可开始有条件地代替一部分特定的脑力劳动, 因而人类已面临另一场更宏伟的技术革命, 处在又一个新时代的前夕. 数学是一种典型的脑力劳动, 它在这一场新的技术革命中, 无疑将扮演一个重要的角色. 为了了解数学在当前这场革命中所扮演的角色, 就应对机器的作用, 以及作为数学的脑力劳动的方式, 进行一定的分析.

1. 什么是数学的机械化

不论是机器代替体力劳动, 或是计算机代替某种脑力劳动, 其所以成为可能, 关键在于所需代替的劳动已经"机械化", 也就是说已实现了刻板化或规格化. 正因为割麦、刈草、纺纱、织布的动作已经是机械化刻板化了的, 因而可据此造出割麦机、刈草机、纺纱机、织布机来. 也正因为加减乘除开方等运算这一类脑力劳动, 几千年来就已经是机械地刻板地进行的, 才有可能使得 17 世纪的法国数学家 Pascal, 利用齿轮传动造出了第一台机械计算机——加法机, 并由 Leibniz 改进成为也能进行乘法的机器. 数学问题的机械化, 就要求在运算或证明过程中, 每前进一步之后, 都有一个确定的、必须选择的下一步, 这样沿着一条有规律的、刻板的道路, 一直达到结论.

在中小学数学的范围里, 就有着不少已经机械化了的课题. 除了四则、开方等运算外, 解线性联立方程组就是一个很好的例子. 在中学用的数学课本中, 往往介绍解线性方程组的各种"消去法", 其求解过程是一个按一定程序进行的计算过程, 也就是一种机械的、刻板的过程. 根据这一过程编成程序, 由电子计算机付诸实施, 就可以不仅机器化而且达到自动化, 在几分钟甚至几秒钟之内求出一个未知数多至上百个的线性方程组的解答来, 这在手工计算几乎是不可能的. 如果

① 20 世纪七八十年代之交, 我尝试用计算机证明几何定理取得成功, 由此提出了数学机械化的设想. 先后在一些通俗报告与写作中, 解释数学机械化的意义与前景, 例如 1978 年发表于《自然辩证法通讯》的"数学机械化问题"以及 1980 年发表于《百科知识》的 "数学的机械化". 二文都重载于 1995 年由山东教育出版社出版的《吴文俊论数学机械化》一书. 经过 20 多年众多学者的努力, 数学机械化在各个方面都取得了丰富多彩的成就, 并已出版了多种专著, 汇集成现在的数学机械化丛书. 现据 1980 年的《百科知识》的 "数学的机械化" 一文, 稍加修改并作增补, 以代丛书前言.

用手工计算，即使是解只有三四个未知数的方程组，也将是繁琐而令人厌烦的. 现代化的国防、经济建设中，大量出现的例如网络一类的问题，往往可归结为求解很多未知数的线性方程组. 这使得已经机械化了的线性方程解法在四个现代化中起着一种重要作用.

即使是不专门研究数学的人们，也大都知道，数学的脑力劳动有两种主要形式：数值计算与定理证明(或许还应包括公式推导，但这终究是次要的). 著名的数理逻辑学家美国洛克菲勒大学教授王浩先生在一篇有名的《向机械化数学前进》的文章中，曾列举了这两种数学脑力劳动的若干不同之点. 我们可以简略而概括地把它们对比一下：

计算	证明
易	难
繁	简
刻板	灵活
枯燥	美妙

计算，如已经提到过的加、减、乘、除、开方与解线性方程组，其所以虽繁而易，根本原因正在于它已经机械化. 而证明的巧而难，是大家都深有体会的，其根本原因也正在于它并没有机械化. 例如，我们在中学初等几何定理的证明中，就经常要依靠诸如直观、洞察、经验、以及其他一些模糊不清的原则，去寻找捷径.

2. 从证明的机械化到机器证明

一个值得提出的问题是：定理的证明是不是也能像计算那样机械化，因而把巧而难的证明，化为计算那样虽繁而易的劳动呢？事实上，这一证明机械化的设想，并不始自今日，它早就为 17 世纪时的大哲学家、大思想家和大数学家 Descates 和 Leibniz 所具有. 只是直到 19 世纪末，Hilbert(德国数学家，1862~1943)等创立并发展了数理逻辑以来，这一设想才有了明确的数学形式. 又由于20世纪40年代电子计算机的出现，才使这一设想的实现有了现实可能性.

从 20 世纪二三十年代以来，数理逻辑学家们对于定理证明机械化的可能性进行了大量的理论探讨，他们的结果大都是否定的. 例如 Gödel 等的一条著名定理就说，即使看来最简单的初等数论这一范围，它的定理证明的机械化也是不可能的. 另一面，1950 年波兰数学家 Tarski 则证明了初等几何(以及初等代数)这一范围的定理证明，却是可以机械化的. 只是 Tarski 的结果近于例外，在初等几何及初等代数以外的大量结果都是反面的，即机械化是不可能的. 1956 年以来美国开始了利用电子计算机做证明定理的尝试. 1959 年王浩先生设计了一个机械化方法，用计算机证明了 Russell 等著的《数学原理》这一经典著作中的几百条定理，只用了9分钟，在数学与数理逻辑学界引起了轰动. 一时间，机器证明的前景似乎非

常乐观. 例如 1958 年时就有人曾经预测：在 10 年之内计算机将发现并证明一个重要的数学新定理. 还有人认为, 如果这样, 则不仅许多著名哲学家与数学家如 Peano、Whithead、Russell、Hilbert 以及 Turing 等人的梦想得以实现, 而且计算将成为科学的皇后, 人类的主人!

然而, 事情的发展却并不如预期那样美好. 尽管在 1976 年, 美国的 Hanker 等人, 在高速计算机上用了 1200 小时的计算时间, 解决了数学家们 100 多年来所未能解决的一个著名难题——四色问题, 因此而轰动一时, 但是, 这只能说明计算机作为定理证明的辅助工具有着巨大潜力, 还不能认为这样的证明就是一种真正的机器证明. 用王浩先生的说法, Hanker 等关于四色定理的证明是一种使用计算机的特例机证, 它只适用于四色这一特殊的定理, 这与所谓基础机器证明之能适用于一类定理者有别. 后者才真正体现了机械化定理证明, 进而实现机器证明的实质. 另一面, 在真正的机械化证明方面, 虽然 Tarski 在理论上早已证明了初等几何的定理证明是能机械化的, 还提出了据以造判定机也即是证明机的设想, 但实际上他的机械化方法非常繁, 繁到不可收拾, 因而远远不是切实可行的. 1976 年时, 美国做了许多在计算机上证明定理的实验, 在 Tarski 的初等几何范围内, 用计算机所能证明的只是一些近于同义反复的“儿戏式”的“定理”. 因此, 有些专家曾经发出过这样悲观的论调：如果专依靠机器, 则再过 100 年也未必能证明出多少有意义的新定理来.

3. 一条切实可行的道路

1976 年冬, 我们开始了定理证明机械化的研究. 1977 年春取得了初步成果, 证明初等几何主要一类定理的证明可以机械化. 在理论上说来, 我们的结果已包括在 Tarski 的定理之中. 但与 Tarski 的结果不同, 我们的机械化方法是切实可行的, 即使用手算, 依据机械化的方法逐步进行, 虽然繁复, 也可以证明一些艰深的定理.

我们的方法主要分两步, 第一步是引进坐标, 然后把需证定理中的假设与终结部分都用坐标间的代数关系来表示. 我们所考虑的定理局限于这些代数关系都是多项式等式关系的范围, 例如平行、垂直、相交、距离等关系都是如此. 这一步可以叫做几何的代数化. 第二步是通过代表假设的多项式关系把终结多项式中的坐标逐个消去, 如果消去的结果为零, 即表明定理正确, 否则再作进一步检查. 这一步完全是代数的, 即用多项式的消元法来验证.

上述两步都可以机械与刻板地进行. 根据我们的机械化方法编成程序, 以在计算机上实现机器证明, 并无实质上的困难. 事实上数学所某些同志以及国外的王浩先生都曾在计算机上试行过. 我们自己也曾在国产的长城 203 台式机上证明了像 Simson 线那样不算简单的定理. 1978 年初我们又证明了初等微分几何中主要的一类定理证明也可以机械化. 而且这种机械化方法也是切实可行的, 并据此用

手算证明了不算简单的一些定理.

从我们的工作中可以看出, 定理的机械化证明, 往往极度繁复, 与通常既简且妙的证明形成对照, 这种以量的复杂来换取质的困难, 正是利用计算机所需要的.

在电子计算机如此发展的今天, 把我们的机械化方法在计算机上实现不仅不难, 而且有一台微型的台式机也就够了. 就像我们曾经使用过的长城 203, 它的存数最多只能到 234 个 10 进位的 12 位数, 就已能用以证明 Simon 线那样的定理. 随着超大规模集成电路与其他技术的出现与改进, 微型机将愈来愈小型化而内存却愈来愈大, 功能愈来愈多, 自动化的程度也愈来愈高. 进入 21 世纪以后, 这一类方便的小型机器将为广大群众普遍使用. 它们不仅将成为证明一些不很简单的定理的武器, 而且还可用以发现并证明一些艰深的定理, 而这种定理的发现与证明, 在数学研究手工业式的过去, 将是不可想象的. 这里我们应该着重指出, 我们并不鼓励以后人们将使用计算机来证明甚至发现一些有趣的几何定理. 恰恰相反, 我们希望人们不再从事这种虽然有趣却即是对数学甚至几何学本身也已意义不大的工作, 而把自己从这种工作中解放出来, 把自己的聪明才智与创造能力贯注到更有意义的脑力劳动上去.

还应该指出, 目前我们所能证明的定理, 局限于已经发现的机械化方法的范围, 例如初等几何与初等微分几何之内. 而如何超出与扩大这些机械化的范围, 则是今后需要探索的长期的理论性工作.

4. 历史的启示与中国古代数学

我们发现几何定理证明的机械化方法是在 1976 至 1977 年之间. 约在两年之后我们发现早在 1899 年出版的 Hilbert 的经典名著《几何基础》中, 就有着一条真正的正面的机械化定理：初等几何中只涉及从属与平行关系的定理证明可以机械化. 当然, 原来的叙述并不是以机械化的语言来表达的, 也许就连 Hilbert 本人也并没有对这一定理的机械化意义有明确的认识, 自然更不见得有其他人提到过这一定理的机械化内容. Hilbert 是以公理化的典范而著称于世的, 但我认为, 该书更重要处, 是在于提供了一条从公理化出发, 通过代数化以到达机械化的道路. 自然, 处于 Hilbert 以及其后数学的一张纸一支笔的手工作业时代里, 公理化的思想与方法得到足够的重视与充分的发展, 而机械化的方向与意义受到数学家的忽视是完全可以理解的. 但电子计算机已日益普及, 因而繁琐而重复的计算已成为不足道的事情, 机械化的思想应比公理化思想受到更大重视, 似乎是合乎实际的.

其次应该着重指出, 我们从事机械化定理证明工作获得成果之前, 对 Tarski 的已有工作并无接触, 更没有想到 Hilbert 的《几何基础》会与机械化有任何关系. 我们是在中国古代数学的启发之下提出问题并想出解决办法来的.

说起来道理也很简单：中国的古代数学基本上是一种机械化的数学. 四则运算与开方的机械化算法由来已久. 汉初完成的《九章算术》中, 对开平、立方与解线性联立方程组的机械化过程, 都有详细说明. 宋代更发展到高次代数方程求数值解的机械化算法.

总之, 各个数学领域都有定理证明的问题, 并不限于初等几何或微分几何. 这种定理证明肇始于古希腊的 Euclid 传统, 现已成为近代纯粹数学或核心数学的主流. 与之相异, 中国的古代学者重视的是各种问题特别是来自实际要求的具体问题的解决. 各种问题的已知数据与要求的数据之间, 很自然地往往以多项式方程的形式出现. 因之, 多项式方程的求解问题, 也就自然成为中国古代数学家研究的中心问题. 从秦汉以来, 所研究的方程由简到繁, 不断有所前进, 有所创新. 到宋元时期, 更出现了一个思想与方法的飞跃：天元术的创立.

“天元术”到元代朱世杰时又发展成四元术, 所引入的天元、地元、人元、物元实际上相当于近代的未知元或未知数. 将这些未知元作为通常的已知数那样加减乘除, 就可得到与近代多项式与有理函数相当的概念与相应的表达形式与运算法则. 一些几何性质与关系很容易转化成这种多项式或有理函数的形式及其关系. 这使得过去依题意列方程这种无法可循需要高度技巧的工作从此变成轻而易举. 朱世杰 1303 年的《四元玉鉴》又给出了解任意多至四个未知元的多项式方程组的方法. 这里限于 4 个未知元只是由于所使用的计算工具(算筹和算板)的限制. 实质上他解方程的思想路线与方法完全可以适用于任意多的未知元.

不问可知, 在当时的具体条件下, 朱世杰的方法有许多缺陷. 首先, 当时还没有复数的概念, 因之朱世杰往往限于求出(正)实值. 这无可厚非, 甚至在 17 世纪 Descartes 的时代也还往往如此. 但此外朱世杰在方法上也未臻完善. 尽管如此, 朱世杰的思想路线与方法步骤是完全正确的, 我们在 20 世纪 70 年代之末, 遵循朱世杰的思想路线与方法的基本实质, 采用美国数学家 J.F.Ritt 在 1932, 1950 年关于微分方程代数研究书中所提供的某些技术, 得出了解任意复多项式方程组的一般算法, 并给出了全部复数解的具体表达形式. 此后又得出了实系数时求实解的方法, 为重要的优化问题提供了一个具体的方法.

由于多种问题往往自然导致多项式方程组的求解, 因而我们解方程的一般方法可被应用于形形式式的问题. 这些问题可以来自数学自身, 也可以来自其他自然科学或工程技术. 在本丛书的第一本书, 吴文俊的《数学机械化》一书中, 可以看到这些应用的实例. 在工程技术方面的应用, 在本丛书中已有高小山的《几何自动作图与智能 CAD》与陈发来和冯玉瑜的《代数曲面拼接》两本专著. 上述解多项式方程组的一般方法已推广至代微分方程的情形. 许多应用以及相应论著正在酝酿之中.

5. 未来的技术革命与时代的使命

宋元时代天元术与四元术的创造, 把许多问题特别是几何问题转化成代数方程与方程组的求解问题. 这一方法用于几何可称为几何的代数化. 12 世纪的刘益将新法与“古法”比较, 称“省功数倍”, 这可以说是减轻脑力劳动使数学走上机械化的道路的一项伟大的成就.

与天元术的创造相伴, 宋元时代的数学又引进了相当于现代多项式的概念, 建立了多项式的运算法则和消元法的有关代数工具, 使几何代数化的方法得到了系统的发展, 见于宋元时代幸以保存至今的杨辉、李冶、朱世杰的许多著作之中. 几何的代数化是解析几何的前身, 这些创造使我国古代数学达到了又一个高峰. 可以说, 当时我国已到达了解析几何与微积分的大门, 具备了创立这些数学关键领域的条件, 但是各种原因使我们数学的雄伟步伐就在这些大门之前停顿下来. 几百年的停顿, 使我们这个古代的数学大国在近代变成了数学上的纯粹入超国家. 然而, 我国古代机械化与代数化的光辉思想和伟大成就是无法磨灭的. 本人关于数学机械化的研究工作, 就是在这些思想与成就启发之下的产物, 它是我国自《九章算术》以迄宋元时期数学的直接继承.

恩格斯曾经指出, 枪炮的出现消除了体力上的差别, 使中世纪的骑士阶级从此销声匿迹, 为欧洲从封建时代进入到资本主义时代准备了条件. 近年有些计算机科学家指出, 个人用计算机的出现, 其冲击作用可与枪炮的出现相比. 枪炮使人们在体力上难分强弱, 而个人用计算机将使人们在智力上难分聪明愚鲁. 又有人对数学的未来提出看法, 认为计算机的出现, 将使数学现在一张纸一支笔的方法, 在历史的长河中, 无异于石器时代的手工方法. 今天的数学家们, 不得不面对计算机的挑战, 但是, 也不必妄自菲薄. 大量繁复的事情交给计算机去做了, 人脑将仍然从事富有创造性的劳动.

我国在体力劳动的机械化革命中曾经掉队, 以致造成现在的落后状态. 在当前新的一场脑力劳动的机械化革命中, 我们不能重蹈覆辙. 数学是一种典型的脑力劳动, 它的机械化有着许多其他类型脑力劳动所不及的有利条件. 它的发扬与实现对我国的数学家是一种时代的使命. 我国古代数学的光辉, 鼓舞着我们为实现数学的机械化, 在某种意义上也可以说是真正的现代化而勇往直前.

吴文俊
2002 年 6 月于北京

序　　言

并发系统大量存在于现实生活中，如网络系统、通信系统、电路系统等. 由于并发系统的复杂性，对它的研究通常采用的是形式化方法，用具有严格定义的数学模型对其进行刻画、分析和推理. 这方面较为系统的研究工作开始于 20 世纪 80 年代初，研究的重点在于对并发系统的功能特征进行刻画和分析，主要关注系统的行为特征，它能做什么，怎么做. 而性能分析的理论与方法主要是通过对系统各种性能指标特征的研究，确定它到底做得如何. 随着现实中并发系统复杂程度的不断增加，系统的功能特征与性能特征变得密不可分，很难单纯只从功能方面来刻画和描述系统的性质，于是，与并发系统性能分析相关的研究变得越来越重要，由此产生了很多既能对系统进行功能刻画和验证，又可以对系统进行性能描述和评价的形式模型和分析技术.

交互式马尔可夫链 (IMC) 是一种功能行为与性能指标混合的并发系统模型，通过正交结合经典的进程代数模型和连续时间马尔可夫链（CTMC）模型，IMC 提供了完美的可组合化的并发系统的设计及分析框架. 本书主要介绍作者在 IMC 理论与应用方面的系统工作，理论方面介绍了 IMC 模型及 IMC 上的各种分支时间等价和前序关系，应用方面介绍了基于 IMC 的并发系统层次化设计以及功能验证与性能评价方法. 具体包括以下三方面的内容：

- IMC 上的分支时间等价和前序关系. 本书将经典的并发系统功能模型上的强（互）模拟和弱（互）模拟与性能模型上的强（互）模拟和弱（互）模拟概念在统一的框架下进行了定义，同时以基于动作的连续随机逻辑 aCSL 为基础，研究了这些分支时间等价和前序关系的逻辑特征，即它们与逻辑等价性之间的联系. 通过等价关系之间相互联系的研究，得到了 IMC 上的分支时间等价关系谱，它基本上包含了该领域所有功能模型和性能模型上的相关研究结果.
- IMC 上的动作细化理论. 动作细化是经典的并发系统层次化设计方法论的基础，本书介绍了 IMC 上的动作细化理论. 对动作细化采用算子的观点，分别在 IMC 的语法和语义层次上定义了动作细化操作，并且证明了在语义层次上，即 IMC 模型上定义的动作细化具有良好的性质，两种常用的等价 —— 线性时间的交织迹等价和分支时间的交织互模拟等价在动作细化操作之下都是同余的. 而在进程代数 IMPA 语法层次上，证明了上述两种等价关系在安全动作细化下同余.
- 基于 IMC 的模型检验. IMC 的主要目的之一是提供可组合化的功能验证与性能评价模型，本书给出了 IMC 上的模型检验算法，以提供一种统一高效机械化的并发系统的功能验证和性能评价手段. 该算法采用 aCSL 作为系统性质描述语言，完全基于动作，从而可以充分利用 IMC 强大的组合建模能力，达到

对并发系统进行机械化自动验证和评价的目的. IMC 上的模型检验算法是纯功能模型标记转移系统 (LTS) 和纯性能模型连续时间马尔可夫链 (CTMC) 上的模型检验算法的拓展, 并且与已有的算法相一致, 即当 IMC 退化成为 LTS 和 CTMC 时, 该算法也退化成为 LTS 和 CTMC 上的模型检验算法. 作为试验, 用实例演示了如何应用 IMC 上的模型检验算法对基于动作的并发系统进行自动验证和评价.

致 谢

衷心感谢中国科学院数学与系统科学研究院的吴文俊院士, 是他将我带入了数学与计算机科学的交叉领域, 他关于数学与计算机科学的深邃思想与见解使我对理论计算机科学产生了浓厚的兴趣. 感谢已故的北京大学程民德院士, 在他的指导下, 我对信息与计算机科学有了更加深刻的认识. 同时感谢已故的德国 Max-Planck 计算机科学研究所 Harald Ganzinger 教授, 我在 Max-Planck 度过了两年美好的时光, 每天中午午饭后的咖啡屋聚会使我在逻辑学方面受益匪浅. 相信本书是对程民德教授和 Harald Ganzinger 教授的最好怀念.

感谢德国 Mannheim 大学 Mila Majster-Cederbaum 教授, 我和她在 Mannheim 大学长达六年的形式化方法领域的合作研究是本书的基础. 感谢德国 Mannheim 大学 Verina Wolf 博士、Frank Sagler 博士、Kiel 大学 Harald Fecher 博士以及中国科学院软件研究所詹乃军博士, 我的研究得益于与他们的关于形式刻画与性能评价的大量讨论. 感谢美国 Portland 州立大学的宋晓宇教授, 他为本书的写作提供了大量的形式化验证方面的文献.

感谢中国科学院成都计算机应用研究所张景中院士、杨路研究员、中国科学院数学与系统科学研究院高小山研究员、刘卓军研究员以及清华大学应明生教授多年来的支持和帮助.

科学出版社编辑提出了许多中肯的修改意见, 使本书增色不少, 在此表示感谢. 对所有审阅本书以及在本书写作和出版过程中给予热情帮助和支持的同事、家人和朋友表示感谢.

吴尽昭

2006 年 6 月

目　录

第一章 绪　论

1.1 研究背景

并发性 (concurrency) 是现实生活中普遍存在的现象, 包括并行性 (parallel) 和分布性 (distribution). 事实上, 现实中的绝大多数系统在本质上都具有并行性或分布性, 纯顺序的系统很少, 这样的系统称为并发系统 (concurrent systems), 例如并行计算机系统、计算机网络系统、移动通信系统、通信协议、集成电路系统等等. 研究并发系统的行为是最近二十多年理论计算机科学中的一个重要课题, 由于并发系统的复杂性, 对它的研究, 人们通常采用的都是形式化方法 (formal methods). 形式化方法是基于严格数学框架的一种对系统进行刻画、分析和推理的方法, 它通过对系统进行严格的语法和语义定义, 使得系统的分析和推理能够精确化、数学化, 从而保证系统刻画和分析的正确性.

目前, 在并发系统的理论研究领域, 普遍采用的一种形式化方法是进程代数 (process algebra) 方法. 进程代数是一种抽象描述语言, 它提供了用于并发系统建模的严格的数学框架. 在进程代数方法中, 系统的行为用进程来描述, 进程由系统所能执行的动作所组成, 这里, 动作是一个抽象概念, 它的含义不需要具体解释, 用来表示一个抽象的活动或行为, 例如发送一条消息、接收一条消息等等. 因此, 以进程代数为框架的并发系统分析都是建立在动作这个基本概念上的. 以动作为基础, 进程代数用算子的形式定义了进程间的各种相互作用, 例如前缀操作 (.)、顺序操作 (;)、选择操作 (+)、并发操作 (||) 等, 这样, 进程代数就能以一种组合化的方式来描述复杂并发系统的行为, 这种组合化的系统描述方式将一个大的系统看作很多小的子系统组合而成, 所以特别适合于描述大规模复杂并发系统, 进程代数也因此成为了当今复杂并发系统形式刻画与分析的主要工具. 比较著名的进程代数有 R. Milner 提出的 CCS (calculus of communication systems) [79,80], Hoare C A R 提出的 CSP (communicating sequential processes) [28,30], 以及其后在这两种进程代数的基础上由国际标准化组织 (ISO) 标准化的 LOTOS (language of temporal ordering specifications) [64].

并发系统的研究可以从很多不同的角度进行, 但所有的研究都集中在两方面, 一方面是研究如何对系统的功能特征进行刻画分析, 另一方面是研究如何对系统的性能特征进行刻画分析. 另外, 在系统的设计开发过程中, 层次化设计技术是一种非常重要的设计方法, 因此并发系统的层次化设计分析技术也是这个领域中的重要研究内容.

1.1.1 并发系统的功能分析

形式化方法的一个重要应用就是对（并发）系统功能方面的特性进行刻画和验证分析. 所谓的功能特征主要指的是系统“能做什么”和“怎么做”，例如，一个网络协议主要完成两个通信主体之间的信息交换，对它的功能特征进行分析，所关心的就是这个协议主要执行的动作以及这些动作之间的相互联系和作用. 这是最初并发系统的研究所主要关注的问题. 这方面的研究工作导致了最早的一批进程代数如 CCS，CSP 的产生.

进程代数作为并发系统功能行为的一种描述语言，语义问题是它的核心研究内容. 语义研究的目的是建立一个精确的、无歧义的框架，以便对并发系统进行推理和分析. 在各种进程代数的语义研究中，可以将其划分为两大类，一类是所谓的交织（interleaving）语义，一类是所谓的非交织（noninterleaving）或真并发（true concurrency）语义. 这两类语义划分的主要依据是对进程代数中并发算子的不同解释，在交织语义下，两个进程并发执行的含义是两个进程的动作任意交叉执行，从而一个系统的执行动作总是形成一个全序（total-order），在进程代数语言中，这种语义使得并发算子可以用顺序算子和选择算子来表示，如

$$a||b = a;b + b;a.$$

这种形式的转换在进程代数理论中称为展开律（expansion law），它在某些情况下可以使得我们对进程行为的推理和分析更加方便. 相反，真并发语义对并发算子不是以两个进程的动作任意交叉执行来解释，而是保留了并发的真正含义，即两个并发执行的动作可以是同时的，它们之间没有先后顺序关系，从而在一个系统的执行中，其执行动作之间不存在全序关系，而是存在偏序关系，因此这种语义又叫做偏序语义. 在这种语义框架下，进程代数语言中没有展开律.

用进程代数研究并发系统的另一个重点是进程的功能行为等价性（behaviour equivalence）问题，即不同的进程描述在什么意义下行为是等价的，可以看成相同的，从而使得在进程代数理论框架下可以建立各种行为相等或不等性公式，进而对进程的功能行为进行推理和分析. 这种等价性问题最早由 Milner R 在 CCS 中进行了研究，并且提出了两类基本的等价关系，一种是线性时间（linear time）的迹等价（trace equivalence），一种是分支时间（branching time）的互模拟等价（bisimulation equivalence）. 这些功能行为等价关系在随后的研究中得到了不断细化，其中以 van Glabbeek R J 的研究最为值得关注，他将各种行为等价关系从线性时间到分支时间进行分类，定义了 11 种不同的等价关系，建立了一个等价关系谱，并研究了这些等价关系的性质以及它们之间的相互联系. 这些等价关系的研究使得人们对并发系统的功能行为有了更深刻和细致的认识.

1.1.2 并发系统的性能分析

并发系统性能分析所关注的是并发系统性能上的特征，例如时间特征、概率特征、随机特征等. 在传统的性能评价领域，人们已经建立了不少成熟的性能评价模型、如排队模型、马尔可夫模型等. 通常这些性能模型并不关心和涉及系统功能方面的特性，而仅仅是从性能方面来刻画系统，由此就产生了将系统的功能分析与性能分析完全割裂开来的两个不同的领域，两个领域的研究者分别采用各自的模型和方法来研究各自领域的问题，导致了大量相互无关的模型与分析技术的产生.

然而，随着现代系统复杂度的增加，尤其是并发系统的大量存在，现代复杂并发系统的功能特征与性能特征之间的界限已经越来越模糊，很多系统性能方面的特征与系统的功能特征往往是密切相关的，因此单纯的功能模型或性能模型都无法有效地对这样的复杂系统进行刻画和分析，这就促使人们开始研究如何将系统的功能分析与性能分析有效地结合在一起，形成了最近十年来并发系统研究领域的另一个重点方向，很多集功能和性能分析于一体的模型被提了出来，成为研究这一交叉领域的有力工具.

对并发系统进行性能分析，通常采用的方法是在原有的功能模型的基础上加入性能数量指标，使得到的模型既能描述系统的行为，又能反映某些特定的数量上的性能特征. 例如传统的功能模型有自动机、Petri 网、进程代数等，在这些模型上加入时间、概率等数量指标，得到的模型就可以在描述系统功能行为的同时对系统的性能特征进行分析，这样就能得到统一的既能进行功能分析又能进行性能分析的混合模型，这样的模型称为混合性能评价模型. 这方面的研究代表有随机 Petri 网、随机进程代数、交互式马尔可夫链等.

1.1.3 并发系统的层次化设计分析

“自顶向下，逐步细化”是传统顺序系统设计方法学当中很经典并且很成功的一种系统设计与分析方法，它将系统由抽象到具体划分为不同的层次，从最抽象的顶层开始设计，每次只需要关注当前抽象层次所涉及的内容，而不必考虑底层的细节，然后逐步细化当前抽象层次，得到更详细的下一层实现，直到最后所有的抽象行为都被具体可操作的行为所实现为止. 这种方法被称为“层次化设计方法”(hierarchical design methodology)，这种层次化的思想在顺序程序设计当中得到了成功的应用，例如函数、过程调用. 因此，将这种层次化的思想应用到并发系统的设计分析当中就是自然而然的事情了.

在并发系统理论当中，层次化思想是以动作细化 (action refinement) 为核心来实现的. 由于大多数并发系统理论框架都将系统的基本行为定义在一些抽象动作的集合之上，层次化设计思想就是要解决如何将上层的一个抽象动作用更具体的一些动作来替换，从而得到更为具体的下层实现. 动作细化理论就提供了这样一个将

高层次的抽象动作转换到低层次的具体实现的机制.

现在关于进程代数框架下的动作细化概念存在如下几种学术观点：

- **算子和非算子的观点** 动作细化的算子观点类似于顺序程序设计中的过程调用，将动作细化定义为进程代数中的一个操作算子，该算子将每一个需要被细化的动作用它的具体实现来代替，这种观点的动作细化主要研究的是算子的同余性问题 (congruence problem)，即找到一个等价关系，它在动作细化算子下是保持的. 而动作细化的非算子观点则不是将动作细化作为进程代数中的一个算子，而是将它看作一种实现关系，这种实现关系用于正确关联不同抽象层次的系统描述.
- **原子和非原子的观点** 根据动作执行的可中断性，动作细化还有原子和非原子的两种不同观点. 原子性的动作细化将动作的执行看作一个不可分割的整体，其执行过程不允许被其他动作中断，而非原子性的动作细化则没有这样的限制，允许一个动作的执行被其他动作所中断. 在实际中，动作的原子性和非原子性观点都有合适的应用场合，例如，将一种语言用另外一种语言实现的时候，采用原子性观点是这种实现正确性的重要保证，而对于两个互相并发执行的动作，它们之间没有任何制约，这时候两个动作的执行采用非原子的观点则更符合实际情况. 因此这两种观点没有好坏之分，采用什么观点是因具体应用而异的.
- **语法和语义的观点** 由于并发系统既可以用语言在语法层次上描述，也可以用模型在语义层次上描述，相应地，动作细化也可以在语法和语义两个层次上分别解释. 这就是动作细化的语法和语义观点的不同. 语法上的动作细化在语言层次进行，如同程序设计语言中的过程调用语句，只考虑它的形式，概念上非常简单，因此适合于实际的使用，而语义上的动作细化则主要在系统的语义模型上进行，它使得语法上的动作细化概念更加精确. 由于语法动作细化可能产生抽象层次上的混淆，语法和语义下的动作细化通常是不一致的，因此语法和语义动作细化的研究要解决这两种细化在什么条件什么意义下一致.

1.2 研究内容

本书的主要研究对象是交互式马尔可夫链 (IMC)，它是一种既能用于并发系统的功能刻画，同时又能用于并发系统的性能分析的模型，主要的研究内容包括 IMC 上各种分支时间等价关系的建立以及它们之间的相互联系和性质，基于 IMC 的模型检验和基于 IMC 的动作细化理论.

1.2.1 等价关系

等价关系在并发系统理论研究当中具有重要地位，它在不同的系统行为之间建立了比较的依据，是并发系统理论和应用研究的基础，例如系统状态空间的化简、同余性问题等. 在经典的并发系统理论当中，各种功能行为等价关系的概念都已经建立起来，这些等价关系可以从根本上划分为两类，一类是所谓的线性时间等价关系，如迹等价；另一类是所谓的分支时间等价关系，如互模拟等价. 对于传统的并发系统的功能模型，这些等价关系主要用于比较两个不同系统的行为，并建立各种代数系统的相等公理. 随着现代并发系统复杂性的增加，系统的可靠性等性能特征变得越来越重要，人们就把这些等价关系的概念引入性能模型当中，建立了概率系统模型、随机系统模型上的等价关系，如离散时间马尔可夫链和连续时间马尔可夫链上的互模拟等价关系，在这些性能模型上建立的等价关系是进行系统状态空间化简的重要手段.

本书在已有工作的基础上，将分支时间的等价关系概念定义在 IMC 模型上，系统地研究并建立起 IMC 模型的各种较为复杂的分支时间等价关系，主要阐明了互模拟等价和模拟等价关系以及它们之间的联系，同时本书还对这些等价关系的逻辑特征进行了探讨，从而使得 IMC 模型的等价关系理论更加丰富和完善. 由于 IMC 模型兼具功能行为分析和性能评价两种功能，这些等价关系的定义使得我们既可以对 IMC 模型的行为和性能特征进行比较，还能简化功能和性能分析模型的状态空间，即我们建立的等价关系综合了已有等价关系的功能，是现有的功能验证分析领域和性能评价分析领域分支时间等价的拓广.

1.2.2 模型检验

模型检验是一种并发系统形式化验证技术，它通过将系统表示成某种状态转移图 (state transition graph)，将系统的性质表示成某种时序逻辑 (temperal logic) 公式，并提供一个有效的算法来检验该状态转换图是否满足给定的时序逻辑公式，从而达到验证系统正确性的目的. 这是一种已经被实践证明了的自动高效的形式化验证方法，被广泛应用于包括软件和硬件系统的并发系统验证.

模型检验最早用于系统的功能性验证，典型的描述系统功能性质的逻辑有计算树逻辑 CTL (computation tree logic) 及 CTL*、线性时间逻辑 LTL (linear time logic) 等，相应的模型检验算法有 CTL 模型检验算法、LTL 模型检验算法等. 随后，人们又将模型检验技术应用到性能模型上，通过扩展 CTL，使其能够表达和刻画各种性能指标，如概率 CTL (PCTL)、连续随机逻辑 CSL (continuous stochastic logic) 等，由此发展了 PCTL 模型检验、CSL 模型检验技术等，这使得模型检验成为了一种先进的性能评价技术. 由于 IMC 模型结合了传统的功能模型和性能模型，我们可以很自然地将模型检验技术引入进来. 本书将模型检验技术应用到了 IMC

模型上，提出了以模型检验为基础的 IMC 模型的功能验证和性能评价技术. 我们主要研究了基于 IMC 的刻画逻辑 aCSL，建立了相应的模型检验算法，讨论了各种优化技术，并给出了一个以数值迭代为基础的算法实现. 这一研究使得 IMC 模型的功能验证和性能评价能力得到了较大的提高，进一步完善和充实了 IMC 的理论和应用领域.

1.2.3 动作细化

IMC 模型最早是作为一种可以组合化的混合性能评价模型被提出来的，其最大的优点是在刻画系统性能特征的同时还提供了对复杂并发系统的组合描述能力，使得对复杂并发系统的性能评价可以得到简化. 由于 IMC 的组合描述能力也是基于系统的动作表示，也可以将 IMC 看作一个基于动作的并发系统模型，从而可以对其进行层次化理论研究.

IMC 模型与其他基于动作的并发系统模型有很大不同，它结合了系统的性能特征，同时又是一个交织语义模型，我们通过给 IMC 模型增加动作细化操作来实现 IMC 的层次化设计分析能力. 与传统层次化理论的研究重点一样，我们主要研究 IMC 语法和语义两个层次上的动作细化，语法层次采用进程代数语言，通过定义相应的动作细化算子，使得 IMC 可以在进程代数语言上进行层次化描述. 语义层次的动作细化直接在 IMC 模型上进行，它提供了直观的、无歧义的动作细化操作的解释. 通过对 IMC 模型的等价关系研究，我们对语法和语义动作细化的同余性问题和一致性问题进行了研究，得到了比较令人满意的结果.

1.2.4 相关工作

本书内容的涉及面比较广泛，以下列出其中一些主要工作，它们是本书研究的基础.

并发系统的等价关系是以进程代数为框架的并发理论研究中的核心问题，它一方面建立起对并发系统的行为进行比较的准则，另一方面又是建立进程代数中各种代数公理系统的基础. 经典的并发系统下的功能行为等价关系概念由 Milner 在他提出的 CCS 中进行了研究 [79,80]，随后 van Glabbeek R J 将并发系统中的行为等价概念进一步细化，得到了一个从线性时间到分支时间的等价关系谱 [97,100]. 由于分支时间等价关系，尤其是互模拟等价关系，具有良好的性质，因此在许多领域中都得到了广泛的研究. 文献 [65, 69] 最先将分支时间等价关系中的（互）模拟等价概念引到概率模型上，此后，各种概率并发模型和随机并发模型上的互模拟概念都被建立起来 [5,14,16,21,25,62,66,85,93,105,]. 文献 [18] 则在这些工作的基础上将概率和随机模型上的各种分支时间等价关系统一起来，研究了它们之间的相互联系及逻辑特征，得到了一个离散时间与连续时间马尔可夫链上的分支时间等价关系谱. 这些工作形成了本书进行 IMC 上的分支时间等价关系研究的基础和动机.

模型检验作为一种高效机械化的验证技术已经被成功地应用于解决工业上的真正实际问题，它是由 Clarke[31,41] 及 Queille 等 [90] 于 20 世纪 80 年代初分别独立发展起来的，到现在已经形成了一套相对比较完善的理论 [33]. 由于模型检验通常涉及利用某种时序逻辑 [29,40] 来表示系统的特征性质，因此使得基于逻辑层次的并发系统研究得到很大发展，各种反映系统不同特征的时序逻辑被建立起来 [6,12,15,41,42,51,52,68,80]，其中包括很多刻画系统性能指标的逻辑，如 RTCTL、PCTL、CSL 等，受经典模型检验成功应用的影响，基于这些具有性能指标的逻辑的模型检验方法也同时得到了大量研究，模型检验也因此被认为是一种代表了目前先进水平的性能评价技术 [11,13,56], 其中 CTMC 上的模型检验给出了一个纯性能模型上的性能评价方法 [12,15]，即 CSL 模型检验，文献 [57] 则将其扩展到随机进程代数 (SPA) 模型, 文献 [54, 62] 提出了基于动作的逻辑 aCSL, 并研究了其上的模型检验算法，他们的工作也是本书研究 IMC 上的模型检验的主要基础.

动作细化的研究开始于 20 世纪 80 年代末 90 年代初，并且作为并发理论研究中的一个热点问题一直持续到 90 年代中后期. 这期间产生了大量的研究论文和博士论文 [2,36,37,47～49,63,92,98,99,103]，这些研究使得人们对动作细化的概念和本质都有了更加深入的理解. 在这一领域当中，以 Glabbeek，Goltz，Gorrieri 和 Rensink 等人的研究工作最为突出，Glabbeek 和 Goltz 主要在语义层次上研究了动作细化问题，提出了各种等价关系及这些等价语义下的动作细化，并讨论了这些等价关系关于动作细化的同余性问题 [95,96,104]，Gorrieri 和 Rensink 则对语法和语义动作细化做了深入的研究和对比 [46,47]. 近年来，文献 [44, 45, 70, 73 ∼ 75, 89] 在真并发语义下研究了各种带有数量性能指标（如实时、概率、随机时间等）及具有不同特征表示的并发系统的动作细化. 这些研究的思想和成果构成了本书进行 IMC 上的动作细化研究的主要内容和基础.

1.3 本书组织

本书分为六个章节，下面对每一章节的主要内容作一简要概括说明.

- **第二章 预备知识** 本章对本书涉及的数学基础理论作一简要介绍，主要内容包括概率、随机变量、分布函数及随机过程，其中重点介绍了马尔可夫链. 作为具有广泛应用的一种数学模型，马尔可夫链的理论已经比较丰富和成熟，交互式马尔可夫链的性能分析基础就是基于马尔可夫链模型的，因此本章对马尔可夫链的相关性质作了重点介绍.
- **第三章 交互式马尔可夫链** 本章介绍本书的主要研究对象 —— 交互式马尔可夫链 (IMC)，详细说明了 IMC 的优点和选用该模型进行研究的动机. 交互式马尔可夫链是传统的标记转移系统与连续时间马尔可夫链相结合的并发系统模型，其中标记转移系统是经典的进程代数语义模型，而连续时间马尔可夫

夫链则是应用最为广泛的性能评价模型，在结合这两种模型的时候，IMC 采取了正交结合的方式，最大程度地保持了原有两种模型的优点，尤其是它的同步组合操作非常简单和自然，使得 IMC 成为一种完美的可以组合化的性能评价模型. 本章分别从语言、结构和逻辑三个层次对 IMC 模型进行了形式刻画，给出了 IMC 的代数描述语言IMPA和 IMC 的刻画逻辑 aCSL，IMPA 和 aCSL 是本书后续工作的基础.

- **第四章 分支时间等价和前序关系** 本章讨论 IMC 的各种分支时间等价关系. 分支时间等价关系主要分为互模拟和模拟等价关系，分别包括强（互）模拟和弱（互）模拟等价关系，我们将传统的分支时间等价关系的概念拓广到了 IMC 模型之上，分别定义了 IMC 模型上的强（互）模拟和弱（互）模拟等价关系，并且基于 aCSL 研究了这些等价关系的逻辑特征，得到了一个系统的 IMC 模型的分支时间等价关系谱.
- **第五章 动作细化** 本章探讨 IMC 模型的层次化理论，我们采用的观点是将动作细化作为一个操作算子，并且使用非原子的动作细化观点. 我们分别在语法和语义层次上定义了动作细化操作，由于 IMC 是交织语义模型，通常的很多等价关系在动作细化操作下并不保持，即这些等价关系关于动作细化都不是同余的，我们采取的策略是对动作细化操作进行一定的限制，使得通常的等价关系在这个限制的动作细化操作下是同余的. 最后，我们阐述了语法和语义动作细化在什么意义下是一致的. 这样，就在 IMC 模型上建立了完整的层次化理论框架，使得基于 IMC 的并发系统层次化设计分析成为可能.
- **第六章 模型检验** 本章讨论 IMC 模型上的模型检验，建立了 IMC 的模型检验算法. 在 IMC 的模型检验问题中，系统模型用 IMC 表示，系统性质用 aCSL 公式表示，与通常的模型检验不同，IMC 模型检验完全基于动作，这使得我们可以充分利用 IMC 的组合描述能力来验证更大规模的并发系统. 我们给出的模型检验算法综合了系统的功能验证与性能评价方法，是纯功能模型和性能模型上的模型检验算法的拓展，即当 IMC 模型退化为纯功能模型和纯性能模型时，得到的就是纯功能模型的验证和纯性能模型的评价方法. 同时也给出了模型检验算法的一个基于数值迭代的实现，讨论了一些可行的优化策略，并且用两个实例说明了 IMC 模型检验的应用.

第二章　预备知识

本章对书中涉及到的一些基本的数学概念和理论作一简要介绍. 主要内容包括概率论的基本概念和随机过程的基本知识, 其中重点介绍马尔可夫随机过程. 对于作为交互式马尔可夫链的基础之一的连续时间马尔可夫链, 详细介绍了它的各种性质.

2.1　概率、随机变量与分布函数

随机现象大量存在于现实当中, 描述随机现象的基本数学工具是概率论. 本章主要介绍公理化概率论的基本内容.

2.1.1　测度空间与概率空间

一个由研究对象全体构成的集合称为样本空间 (sample space), 记为 Ω, Ω 中的元素称为样本点, 记为 ω.

定义 2.1.1　空间 Ω 上的一个非空子集类 $\mathscr{F}$ 若满足条件:

(1) $A \in \mathscr{F} \Rightarrow \overline{A} \in \mathscr{F}$,

(2) $A_1 \in \mathscr{F}, A_2 \in \mathscr{F} \Rightarrow A_1 \cup A_2 \in \mathscr{F}$. 则称 $\mathscr{F}$ 为一个**域** (field), 又称为一个**代数** (algebra) .

定义 2.1.2　设 $\mathscr{F}$ 为 Ω 上的一个域, 且满足

$$A_j \in \mathscr{F}, \quad j = 1, 2, \cdots \Rightarrow \bigcup_{j=1}^{\infty} A_j \in \mathscr{F},$$

则称 $\mathscr{F}$ 为一个 σ **域**, 或 σ **代数**. σ 域中的元素又称为**事件** (event) .

定义 2.1.3　空间 Ω 与其上的一个 σ 域 $\mathscr{F}$ 构成一个**可测空间** (measurable space), 记作 $\langle \Omega, \mathscr{F} \rangle$.

定义 2.1.4　设 $\langle \Omega, \mathscr{F} \rangle$ 为可测空间, μ 为定义在 $\mathscr{F}$ 上的非负函数, 满足:

(1) $\mu(\varnothing) = 0$,

(2) 若 $A_n \in \mathscr{F}, n = 1, 2, \cdots$, 且 $A_n A_m = \varnothing, n \neq m$, 则

$$\mu\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mu(A_n).$$

则称 μ 为 $\mathscr{F}$ 上的**测度**, 称 $\langle \Omega, \mathscr{F}, \mu \rangle$ 为**测度空间** (measure space) .

定义 2.1.5　若测度空间 $\langle \Omega, \mathscr{F}, P \rangle$ 满足 $P(\Omega) = 1$, 则称该测度空间为**概率空间** (probability space), 称 P 为 $\langle \Omega, \mathscr{F} \rangle$ 上的**概率测度**, 简称**概率**.

在一些文献中，σ 域有时也称为 Borel 域，相应得到的可测空间又称为 Borel 空间，本书将保留这种术语称谓. 由以上定义可知，概率是满足定义 2.1.4 及定义 2.1.5 中三个条件的一个测度函数, 分别称这三个条件为非负性（nonnegativity），可列可加性（countable additivity）和规范性（normalization）.

2.1.2 随机变量及其分布函数

随机变量是用来将随机现象量化描述的一个数学概念，它实质上是从一个概率空间到另一个可测空间的映射（可测函数），在实际当中，通常使用下面更加通俗的定义.

定义 2.1.6 设 Ω 是一个样本空间, X 是定义在 Ω 上的实函数 (可测), 即对任一样本点 $\omega \in \Omega$, $X(\omega)$ 为一实数，则称 X 为一个**随机变量**（random variable）.

随机变量一般用大写字母来表示，如 X, Y, Z 等，随机变量的取值则一般用小写字母表示，如 x, y, z 等. 根据随机变量取值的可能性，可以将随机变量分为三种类型：离散型随机变量、连续型随机变量和混合型随机变量. 离散型随机变量只能在有限或可数无穷多个实数点上取值；连续型随机变量只能在一个或多个非退化的实数区间上连续取值，在单个离散点上的取值为零；混合型随机变量则在某些离散点上取值大于零，而在其他地方是连续取值. 本书只涉及离散型随机变量和连续型随机变量.

离散型随机变量通常用它的分布律来表示. 设离散型随机变量 X 所有可能的取值为 $\{x_k, k = 1, 2, \cdots\}$, X 取各个可能值的概率为

$$P\{X = x_k\} = p_k, \quad k = 1, 2, \cdots.$$

则 $\{(x_k, p_k), k = 1, 2, \cdots\}$ 称为 X 的分布律.

连续型随机变量通常由它的密度函数来刻画. 设连续型随机变量 Y 的取值范围为 I, 则存在一个非负实函数 $f(x)$, 使得对任一区间 (a, b), Y 的取值落入该区间的概率为 $P\{Y \in (a, b)\} = \displaystyle\int_a^b f(x)\,dx$，函数 $f(x)$ 就称为 Y 的概率密度函数，简称密度函数.

离散型随机变量和连续型随机变量都可以用分布函数来统一刻画.

定义 2.1.7 设 X 是一个随机变量, x 是任意实数，函数

$$F(x) = P\{X \leqslant x\}$$

称为 X 的**分布函数**.

分布函数 $F(x)$ 具有以下基本性质：

(1) $0 \leqslant F(x) \leqslant 1$ 且

$$F(-\infty) = \lim_{x \to -\infty} F(x) = 0, \quad F(+\infty) = \lim_{x \to +\infty} F(x) = 1;$$

(2) 单调性：$x < y \Rightarrow F(x) \leqslant F(y)$;

(3) 右连续性：$F(x+0) = F(x)$;

(4) $\forall x_1 < x_2, P\{x_1 < X \leqslant x_2\} = F(x_2) - F(x_1)$.

若离散型随机变量 X 有分布律 $\{(x_k, p_k), k = 1, 2, \cdots\}$，则它的分布函数可表示为

$$F(x) = \sum_{x_k \leqslant x} p_k.$$

若连续型随机变量 Y 的密度函数为 $f(x)$，则它的分布函数可表示为

$$F(x) = \int_{-\infty}^{x} f(t)\, dt.$$

由以上可知，若已知随机变量 X 的分布函数，则 X 落在任一区间上的概率都可以知道，从这个意义上说，分布函数完整地刻画了随机变量的统计规律性. 由于分布函数良好的数学性质，因此成为研究随机现象的一个重要的数学工具.

当同时研究多个随机变量时，这些随机变量之间可能会有相互影响，相互依赖，例如，若 X 和 Y 是两个有关的随机变量，当已知 X 的取值时，我们想知道在此条件下，Y 的概率分布是什么？这就是条件概率分布的概念，条件概率分布通常记为 $P\{Y < y \mid X = x\}$，表示当已知 X 的取值时，Y 的概率分布.

2.2 随 机 过 程

随机模型广泛用于描述随时间而随机变化的现象，很多经典的性能分析模型就是随机模型，如马尔可夫模型. 随机模型的数学模型是随机过程，本节重点介绍马尔可夫随机过程，更具体地说，我们关注的是马尔可夫链模型.

随机过程是一族与参数集合 T 有关的随机变量 $\{X_t \mid t \in T\}$，其中 X_t 定义在同一个概率空间，取值于同一个集合 S. 参数集合 T 通常被看作是时间集合，S 称为状态空间. 对于一个固定的 t，随机变量 X_t 代表了 t 时刻状态空间 S 上的一个随机分布. 随机过程的参数时间集合 T 和状态空间都可以是离散或连续的，若 T 为离散的时间集合，称这样的随机过程为离散时间随机过程，若 T 为连续时间的集合，则称为连续时间随机过程. 为简单起见，以下假设连续时间随机过程的时间参数集合为实数集 $\mathbb{R}$，离散时间随机过程的时间参数集合为自然数集 $\mathbb{N}$（包括 0）.

一个随机过程的例子是某个地方，比如成都，在一段时间内的温度 X_t 的变化情况，这个随机过程的状态空间 S 是一个合理的温度变化范围. 而每个随机变量 X_t 就给出了 t 时刻成都任意可能的温度的概率. 如果考虑每天的平均温度的变化，则时间集合 T 就是离散的，由此就得到一个离散时间随机过程，其中 X_0 表示第一

天的平均温度的概率分布，X_1 表示第二天的平均温度的概率分布，依此类推. 类似的，通过选择连续的时间参数集合，就能描述随时间连续变化的随机过程.

一个随机过程在随时间变化的过程当中，其下一个变化时刻的取值可能与当前时刻的取值和当前时刻以前的取值情况都有关，若一个随机过程的下一个变化时刻的取值只与它当前时刻的取值有关而与过去时刻的取值无关，就称这样的随机过程具有马尔可夫性 (Markov property)，或无记忆性 (memoryless)，用数学的语言描述，马尔可夫性就是对任意时间序列 $t_{n+1} > t_n > t_{n-1} > \cdots > t_0$，都有下式成立

$$\begin{aligned} & P\{X_{t_{n+1}} \leqslant x_{n+1} \mid X_{t_n} = x_n, X_{t_{n-1}} = x_{n-1}, \cdots, X_{t_0} = x_0\} \\ = \ & P\{X_{t_{n+1}} \leqslant x_{n+1} \mid X_{t_n} = x_n\}. \end{aligned} \tag{2.1}$$

对于离散时间随机过程来说，马尔可夫性可以表示得更加简洁, 即对任意的 $t \in \mathbb{N}$，有

$$\begin{aligned} & P\{X_{t+1} \leqslant x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \cdots, X_0 = x_0\} \\ = \ & P\{X_{t+1} \leqslant x_{t+1} \mid X_t = x_t\}. \end{aligned} \tag{2.2}$$

因此，马尔可夫性反映了随机过程将来的行为与它过去的行为无关，仅仅与当前所处的状态有关. 但是需要说明的是，马尔可夫性并不意味着随机过程将来的行为与当前的时刻 t 无关，如果 X_t 的取值与 t 有关，这样的随机过程称为非齐次的 (inhomogeneous)，反之，若 X_t 的取值与 t 无关，则称之为齐次的 (homogeneous)，更精确地，齐次性要求对任意 $t' \geqslant t$，下式成立

$$P\{X_t' \leqslant x' \mid X_t = x\} = P\{X_{t'-t} \leqslant x' \mid X_0 = x\}. \tag{2.3}$$

换句话说，满足齐次性的随机过程，其变化过程只与变化的时间间隔有关而与变化的起始时间点无关，这使得我们可以任意选择时间的原点而不会影响观察的结果. 本书只考虑齐次的情况.

例如，在前面的例子中，如果每天的平均温度不依赖于当天所处的天数 (相对于第一天的观察而言)，则得到的随机过程就是齐次的.

称满足马尔可夫性质的随机过程为马尔可夫过程. 若马尔可夫过程的状态空间是离散集合，则称为马尔可夫链. 本书只考虑状态空间为离散的情况.

总的来说，在本书中涉及的随机过程有三个限制条件，一是满足马尔可夫性，二是齐次性，最后状态空间是离散集合. 这样得到的随机过程是一类简单但应用非常广泛的随机模型，它们可以很好地模拟很多现实的随机系统，并且有很多有效的算法来对它们进行数值分析，因此是一个理想的用于系统性能分析的数学模型.

2.2.1 离散时间马尔可夫链

离散时间马尔可夫链 (DTMC) 是时间取值离散，状态空间也是离散集合的马尔可夫过程. 由状态空间的离散性及齐次性, DTMC 的马尔可夫性可以重新表示为

$$\begin{aligned} & P\{X_{t+1}=s' \mid X_t=s, X_{t-1}=s_{t-1},\cdots,X_0=s_0\} \\ = & P\{X_{t+1}=s' \mid X_t=s\} \\ = & P\{X_1=s' \mid X_0=s\}. \end{aligned}$$

这一公式表明，在一个时间变化间隔 (称之为一个时间步 (time step)) 里，从状态 s 变化到状态 s' 的概率与实际开始观察的时刻是无关的. 这就是 DTMC 在两个状态之间的一步转移概率公式. 所有状态之间的一步转移概率构成一个转移概率矩阵，DTMC 可以由这个矩阵完全决定，由此可以定义 DTMC 如下.

定义 2.2.1 一个**离散时间马尔可夫链** (DTMC) 是一个三元组 $(S,\boldsymbol{P},P_0)$，其中

- S 是状态空间 $\{s_1,s_2,\cdots,s_n\}$ 的集合 (n 通常是有限的)；
- $\boldsymbol{P}: S\times S\mapsto[0,1]$ 是 (一步) 转移概率矩阵；
- P_0 是初始分布.

若给定一个唯一的初始状态，则初始分布可以简化为该初始状态，为简单起见，本书以后的定义假定初始分布都为一个给定的初始状态.

记 $\boldsymbol{P}=(p_{ij})_{n\times n}$，其中 p_{ij} 表示系统在一个时间间隔内从状态 s_i 转移到 s_j 的概率. 由于从任何一个状态出发，在下一个时刻系统必然处于状态 $s_1,s_2,\cdots,s_n$ 中的某一个，因此矩阵 $\boldsymbol{P}$的每一行元素之和为 1，即

$$\sum_{j=1}^{n} p_{ij}=1, \quad i=1,2,\cdots,n.$$

根据 DTMC 的转移概率矩阵，可以用一个状态转换图来描述一个 DTMC，如图 2.1 所示.

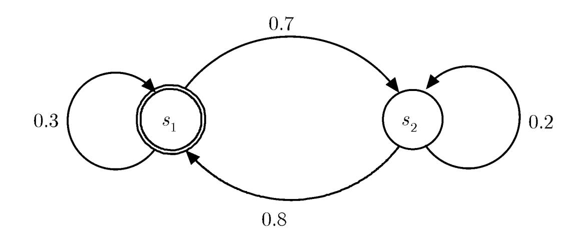


图 2.1 离散时间马尔可夫链图示

其中，状态用圆圈表示，状态名称标在圆圈里面或旁边，状态转移关系用箭头表示，转移概率标在每个箭头上面，初始状态用双线条的圆圈来表示（下同）. 本例中，DTMC 的状态只有两个，分别为 s_1 和 s_2，若系统的初始状态为 s_1，则从 s_1 出发，系统在下一时刻转移到 s_2 的概率是 0.7，保持原来状态不变的概率是 0.3，同样，当系统处于 s_2 状态时，其下一时刻的变化也在图中表示了出来. 注意，由于齐次性，不需要知道系统是何时处于初始状态，以任意时刻作为起点开始观察所看到的系统行为都是一样的.

下面通过这个例子来看看 DTMC 的另外一个重要性质. 在这个例子中，假定系统初始处于状态 s_1，下一个时刻系统仍处于状态 s_1 的概率是 0.3，类似地，再下一个时刻系统仍处于状态 s_1 的概率是 0.3×0.3，由此很容易得出系统在某个状态 s 上的停留时间 ST_s (sojourn time) —— 系统在进入下一个状态所经过的时间步的总数 —— 所满足的概率分布为一个几何分布，即

$$P\{\mathrm{ST}_s = i\} = p^i(1-p),$$

其中 p 是系统在该状态停留一个时间步的概率，如本例中 $p = 0.3$. 值得说明的是这个离散型随机变量的概率分布是无记忆的，这表明在某个状态已经停留的时间信息与剩下的停留时间的概率分布无关，剩下的停留时间所满足的仍然是同一个参数的几何分布. 这个无记忆性实际上就是马尔可夫性的自然结果，既然马尔可夫链将来的行为只依赖于系统当前所处的状态，而与以往所经过的状态无关，在一个状态的停留时间就不会与已经在当前状态停留了多少时间有关了. 回到上面的例子，假设系统已经在状态 s_1 停留了 1000 个时间步，那么系统在状态 s_1 停留整整 1002 个时间步的概率，与系统在状态 s_1 停留 2 个时间步的概率是相等的，即

$$P\{\mathrm{ST}_{s_1} = 1002 \mid \mathrm{ST}_{s_1} > 1000\} = P\{\mathrm{ST}_{s_1} = 2\} = 0.3^2 \times 0.7 = 0.063.$$

有趣的是，几何分布是唯一具有无记忆性的离散型概率分布.

2.2.2 连续时间马尔可夫链

连续时间马尔可夫链 (CTMC) 是随时间连续变化于离散状态空间的马尔可夫过程. 与 DTMC 相比，CTMC 的定义要稍微复杂一些，然而，CTMC 要比 DTMC 应用更加广泛，是大多数性能评价模型的数学基础，因此我们将详细介绍 CTMC 的各个性质.

与 DTMC 一样，先来看 CTMC 中马尔可夫性的表示. 由于 CTMC 的时间参数取值于连续的时间区间，马尔可夫性可以表示为对任意 $t_n + \Delta t > t_n > t_{n-1}$

$> \cdots > t_0$，有

$$\begin{aligned} & P\{X_{t_n+\Delta t} = s' \mid X_{t_n} = s, X_{t_{n-1}} = s_{t_{n-1}}, \cdots, X_0 = s_0\} \\ = \quad & P\{X_{t_n+\Delta t} = s' \mid X_{t_n} = s\} \\ = \quad & P\{X_{\Delta t} = s' \mid X_0 = s\}. \end{aligned}$$

上式最后一个等号是因为齐次性假设. 可以看到，已知系统当前时刻所处的状态，经过 Δt 的时间后，系统进入下一个状态的概率（即转移概率）除了与下一个状态有关，还与时间间隔 Δt 有关，Δt 就是 CTMC 中系统在某个状态上的停留时间，因此上面这个概率实际上也反映了 CTMC 中系统在某个状态 s 的停留时间 ST_s 所满足的概率分布. 与 DTMC 中的情况不同的是，ST_s 在这里是个连续型随机变量. 由于马尔可夫性，ST_s 同样是个无记忆的分布函数，而在连续型随机分布中，只有指数分布是具有无记忆性的，因此可以很容易得出，上面的概率分布可以表示为

$$P\{X_{\Delta t} = s' \mid X_0 = s\} = P\{SJ_s < \Delta t\} = 1 - e^{-\lambda \Delta t},$$

其中，$\lambda \in \mathbb{R}_{\geqslant 0}$ 是指数分布的参数，$\mathbb{R}_{\geqslant 0}$ 是非负实数的集合. 利用 Taylor 展开，又可以得到如下的公式

$$P\{X_{\Delta t} = s' \mid X_0 = s\} = \lambda \Delta t + o(\Delta t),$$

其中 $o(\Delta t)$ 包含了在 Δt 的时间间隔内从状态 s 经过其他状态转移到状态 s' 的概率的总和，它是 Δt 的高阶无穷小，由此可知，在 Δt 的时间间隔内，系统从状态 s 转移到状态 s' 的概率近似正比于 Δt，因此，λ 就反映了系统从一个状态转移到下一个状态的概率随时间变化的比率，正因为如此，参数 λ 也被称为转移率（rate）. 与转移概率不同的是，转移率 λ 与时间间隔长度 Δt 是无关的. 由于每两个状态之间都可能存在这样的转移率，因此，与 DTMC 相似，这些转移率也构成了一个转移率矩阵$\boldsymbol{R}$，CTMC 的状态转移概率就可以由这个转移率矩阵完全描述出来了.

这里隐含了状态 s 与 s' 不相同的假设. 如果 s 与 s' 是同一个状态，则系统在 Δt 的时间间隔里仍然停留在状态 s 的概率 $P\{X_{\Delta t} = s \mid X_0 = s\}$ 随着时间的增加而减少，因此相应的转移率是负值，其含义是由离开状态 s 的概率决定，实际上，它是 s 到其他所有状态的转移率的负数和，由此可知转移率矩阵$\boldsymbol{R}$的每一行的代数和为 0.

通过以上的分析可以知道，CTMC 的随机行为可以完全由初始分布或初始状态与转移率矩阵描述出来，与 DTMC 类似，可以如下定义 CTMC.

定义 2.2.2 **一个连续时间马尔可夫链**（CTMC）是一个三元组 $(S, \boldsymbol{R}, \alpha_0)$，其中

- S 是状态空间 $\{s_1, s_2, \cdots, s_n\}$ 的集合（n 通常是有限的）；

•$\boldsymbol{R}: S \times S \mapsto \mathbb{R}_{\geqslant 0}$ 是转移率矩阵；

•α_0 是初始分布.

定义中最主要的是转移率矩阵 $\boldsymbol{R}$. 记 $\boldsymbol{R}=(\lambda_{ij})_{n\times n}$，由上面的分析，有

$$\lambda_{ii} = -\sum_{j\neq i}^{n} \lambda_{ij}, \quad i = 1, 2, \cdots, n,$$

因此，

$$\sum_{j=1}^{n} \lambda_{ij} = 0, \quad i = 1, 2, \cdots, n.$$

$\boldsymbol{R}$中的元素 λ_{ij} 实际上是指数分布的参数，为了更好地解释这个转移率的含义，有必要先简要介绍一下指数分布的几个重要性质，它们将是今后的章节中很多内容的关键解释.

假设随机变量 D_1 和 D_2 分别满足参数为 λ_1 和 λ_2 的指数分布，即

$$P\{D_1 \leqslant t\} = 1 - e^{-\lambda_1 t}, \quad P\{D_2 \leqslant t\} = 1 - e^{-\lambda_2 t}.$$

则关于指数分布有下面的结论.

性质 1 指数分布由其参数 λ 完全刻画，通常代表一个延迟时间的概率分布，其数学期望（均值）为 $1/\lambda$.

性质 2 指数分布具有无记忆性，即 $P\{D_1 \leqslant t + t_0 \mid D_1 > t_0\} = P\{D_1 \leqslant t\}$.

性质 3 指数分布关于最小值运算封闭，且满足参数为相应指数分布的参数和的指数分布，即

$$P\{\min(D_1, D_2) \leqslant t\} = 1 - e^{(\lambda_1+\lambda_2)t}.$$

性质 4 D_1 小于（大于）D_2 的概率可以直接由相应的指数分布参数得出：

$$P\{D_1 \leqslant D_2\} = \frac{\lambda_1}{\lambda_1 + \lambda_2},$$

$$P\{D_2 \leqslant D_1\} = \frac{\lambda_2}{\lambda_1 + \lambda_2}.$$

性质 5 由于指数分布的连续性，D_1 和 D_2 在同一时刻消逝的概率为 0，即 $P\{D_1 = D_2\} = 0$.

现在可以具体地解释矩阵$\boldsymbol{R}$的含义了. 若 $\lambda_{ij} > 0$，则表示状态 s_i 到 s_j 之间存在一个转移关系，并且根据马尔可夫性，这个状态转移在时间 t 内发生的概率是 $1 - e^{-\lambda_{ij} t}$. 如果存在多个 j，使得 $\lambda_{ij} > 0$，即从状态 s_i 出发有多个可能的转移，则在这些转移之间就存在着所谓的竞争 (race)，竞争的结果是谁最先完成自己的延迟时间就执行谁的状态转移，因此，系统在状态 s_i 的停留时间就是所有这些可能的延迟的最小值，根据指数分布的性质 3，该分布仍然满足指数分布，且分布参数

$\lambda_i = \sum\limits_j \lambda_{ij}$，即所有从状态 s_i 出发的转移率的总和，称之为总转移率 (total rate)，它刻画了系统在状态 s_i 的停留时间所满足的概率，准确地说就是系统在时间 t 内离开状态 s_i 的概率为 $1-e^{-\lambda_i t}$. 如果 $\lambda_i = 0$，则不存在从状态 s_i 出发的转移，称状态 s_i 为吸收态 (absorbing state) . 更进一步，状态 s_j 赢得状态转移竞争，即发生从状态 s_i 到 s_j 的转移的概率为 $p_{ij} = \lambda_{ij}/\lambda_i$，这可以由指数分布的性质 4 得到解释，若 s_i 是吸收态，定义 $p_{ij} = 0, \forall j$. 因此，对于存在竞争的状态 s_i，其在时间 t 内转移到状态 s_j 的概率为

$$\frac{\lambda_{ij}}{\lambda_i} \cdot (1 - e^{-\lambda_i \cdot t}).$$

注意到在上面的式子中，如果令 $t \to +\infty$，可以再次得到从状态 s_i 到 s_j 的转移概率 λ_{ij}/λ_i，这一转移概率实际上刻画了一个与该 CTMC 相关的 DTMC，称之为内嵌的 DTMC (embedded DTMC)，它的转移概率矩阵为$\boldsymbol{P}=(p_{ij})_{n\times n}$.

最后，与 DTMC 类似，可以将 CTMC 用一个状态转移图来直观的表示出来，如图 2.2(a) 所示.

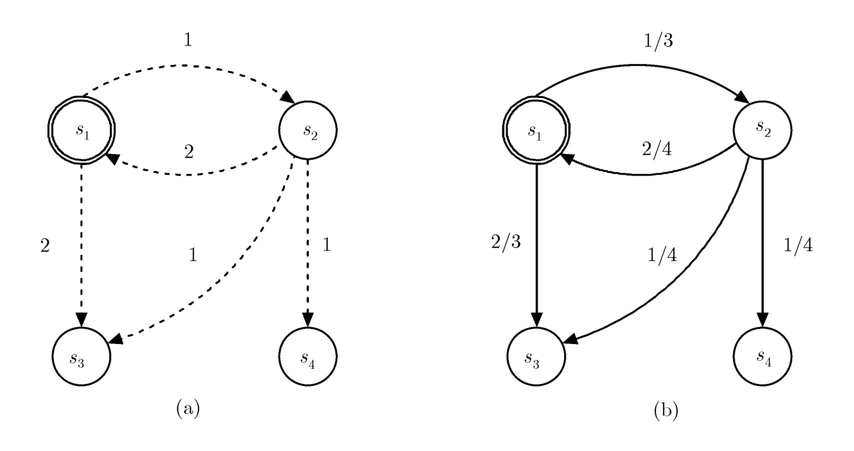


图 2.2　连续时间马尔可夫链及其内嵌的 DTMC

图 2.2(b) 给出了与该 CTMC 相关的内嵌的 DTMC 的图形. 为了与 DTMC 图示中的转移概率相区分，用虚线箭头来表示 CTMC 状态间的转移，上面标上转移率. 我们只标出了转移率 $\lambda_{ij} > 0$ 的状态之间的转移. 例如状态 s_1 到 s_2 的转移率为 1，即在时间 t 内从状态 s_1 转移到 s_2 的概率为 $1 - e^{-1\cdot t}$. 同时从状态 s_1 出发有多个转移可能性，所以在 s_1 上存在竞争条件，系统在 s_1 上停留的时间满足参数为 $1+2=3$ 的指数分布，因此，系统在 2 个时间单位里从状态 s_1 转移到状态 s_2 的概率为

$$\frac{1}{3}(1 - e^{-3\cdot 2}) = 0.3325.$$

2.2.3 马尔可夫分析

以上对马尔可夫链的基本概念和性质作了详细介绍，对于马尔可夫链的分析，已经有很多成熟的理论成果，实际应用中也已经开发了很多有效的算法来对马尔可夫链进行数值分析和计算，这使得马尔可夫链模型在包括物理学、生物学、管理科学、经济学、信息处理以及数字计算方法等各方面都得到了广泛的应用. 在理论计算机科学领域，马尔可夫链就作为一个主要的性能评价模型得到了广泛的研究. 这里简要介绍在性能评价领域里马尔可夫分析主要关心的几个性能度量.

- 到达一个吸收状态的平均时间 MTA；
- 第一次到达某个特定状态 s 的平均时间 MTFP(s)；
- 在一个给定的时刻 t 系统处于一个特定的状态 s 的概率 $P\{X_t = s\}$；
- 系统在长时间运行后稳定在某个状态 s 的概率 $\lim\limits_{t\to\infty} P\{X_t = s\}$（假定该马尔可夫链能达到一个平稳状态）.

当然，以上这些度量不一定对每一个马尔可夫链都有意义，例如，若要计算 MTA，则要求该链必须存在吸收状态且至少有一个吸收态可达才有意义，同样，要计算系统长时间运行稳定后处于某个状态的概率，要求该链必须存在平稳状态.

对于一个齐次马尔可夫链（离散时间或连续时间），如果系统的状态空间是有限的，并且所有状态之间均相互可达（不存在吸收态），则该马尔可夫链存在一个唯一的极限分布，即它能达到一个平稳状态.

平稳分布的计算通常称为稳态分析（steady-state analysis），相对的，计算某一时刻系统处于某个状态的概率 $P\{X_t = s\}$ 称为瞬时分析（transient analysis）. 瞬时分析的计算复杂性相对稳态分析的计算要高，但是适用于任何形状的马尔可夫链. 这些概率的计算都是基于状态的，即它们都是与某个状态有关的概率. 对于稳态分析和静态分析的状态概率的计算，这里只给出 CTMC 的一些相关结论，详细的推导过程读者可以参阅相关文献 [94].

设 $MC = (S, \boldsymbol{R}, \alpha_0)$ 为一个齐次 CTMC，其状态空间是有限的，从状态 s_i 出发，经过时间 t 后，系统处于状态 s_j 的概率

$$p_{ij}(t) = P\{X_t = s_j \mid X_0 = s_i\}.$$

令矩阵 $\boldsymbol{P}(t) = (p_{ij}(t))_{n\times n}$，则该矩阵满足下面的方程（称为 Chapman-Kolmogorov 向前方程）

$$\boldsymbol{P}'(t) = \boldsymbol{P}(t) \cdot \boldsymbol{R}, \tag{2.4}$$

其中 $\boldsymbol{P}(0)$ 满足给定的初始分布.

方程 (2.4) 给出了 CTMC 的瞬时概率满足的数学条件，它实际上是一组微分方程，很多有效的数值方法已经被提了出来，例如龙格 - 库塔方法（Runge-Kutta method），另外，基于归一化（uniformisation）的方法也是一个很有效的 CTMC 瞬

时分析技术，这些成熟的技术和方法成为解决 CTMC 瞬时分析问题的有力工具.

若极限 $\pi_j = \lim\limits_{t\to\infty} p_{ij}(t)$ 存在且不依赖于 i，$\sum\limits_j \pi_j = 1$，则称 $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_n)$ 为该 CTMC 的极限分布（或称平稳分布），该极限分布满足以下方程

$$\boldsymbol{\pi} = \boldsymbol{\pi}\boldsymbol{P}(t), \quad \forall t \geqslant 0,$$

且 $\sum\limits_j \pi_j = 1, \pi_j \geqslant 0$.

利用转移率矩阵 $\boldsymbol{R}$，上式可简化为

$$\boldsymbol{\pi}\boldsymbol{R} = \boldsymbol{0}, \tag{2.5}$$

并且满足 $\sum\limits_j \pi_j = 1, \pi_j \geqslant 0$.

方程 (2.5) 是一组线性方程组，对它的求解有很多经典算法，如基于直接计算的高斯消去法、LU 分解法和基于迭代计算的雅可比迭代、高斯 - 塞德尔迭代等. 因此，CTMC 的稳态分析也具有很成熟的理论和技术上的支持.

由以上的介绍可以看出，马尔可夫链模型具有非常成熟的理论基础，并且与其相关的各种分析和计算也已经（并且仍然在继续地）得到广泛而深入的研究. 另一方面，现实中的很多随机系统都可以用马尔可夫链来模拟，因此马尔可夫链在各个领域都得到了广泛应用. 这也是在并发系统的性能评价领域采用马尔可夫链作为性能模型的一个主要原因.

第三章　交互式马尔可夫链

本章详细介绍我们的主要研究对象 —— 交互式马尔可夫链（interactive Markov chains, IMC）. IMC 作为经典的进程代数模型和连续时间马尔可夫链模型的结合，由德国学者 H. Hermanns 于 1998 年提出 [53], 其目的是提供一种可以组合化的性能评价模型. 进程代数是一个经典的对并发系统进行组合化分析的理论框架, 而连续时间马尔可夫链则是经典的性能评价模型, 将这两者结合起来有不同的方法, 相应地就有不同的模型，而 IMC 将这两者结合的方式是尽可能地保持两者之间的正交性. 本章将详细讨论 IMC 模型的各个组成成分，以及与其他模型相比的优越性. 我们将从结构、语言和逻辑三个层次给出 IMC 的形式刻画，建立完善的 IMC 理论框架，为后续章节的研究作必要的准备.

本章首先从 IMC 的结构层次分别讨论组成 IMC 的两种主要模型 —— 标记转移系统和连续时间马尔可夫链，通过对这两种模型的讨论, 可以更清楚地领会 IMC 所采取的结合两种模型的方式的优点. 然后详细介绍 IMC 模型的组合化操作, 这是 IMC 模型优越性的主要体现. IMC 在语言层次上的描述采用的是进程代数语言 IMPA, 它是经典进程代数语言的一个超集. 在逻辑层次上, 我们定义了一种基于动作的连续随机逻辑 aCSL 来刻画 IMC 的时序性质, 它使得我们可以在 IMC 上进行模型检验, 这部分内容将在第六章中详细阐述.

3.1　进程代数与标记转移系统

本节先简要介绍一下基本的进程代数理论，然后引入标记转移系统.　进程代数是一种抽象描述语言, 它通过一种组合化的方式来描述并发系统, 即将一个大的系统看作是由很多小的子系统构成的, 每个子系统又可以由更小的系统组合而成, 通过进程代数提供的各种组合操作, 可以将一个大系统分解为更小的系统来刻画, 大大降低系统刻画和分析的复杂度. 因此, 这种组合化的思想是目前人们解决复杂并发系统问题的一个主要思路. 例如, 面向对象的程序设计思想, 基于组件的软件设计技术等等都是这一思想的具体体现.

如第一章引言所述, 进程代数有很多具体的系统，如 CCS,CSP,LOTOS 等，这里采用的是一种简化的 LOTOS 进程代数模型，它是作为开放系统互连模型（OSI）的形式刻画语言由国际标准化组织 (ISO) 提出来的, 本书只涉及 LOTOS 的基本语言部分, 有关 LOTOS 的完整介绍读者可以参考文献 [23, 64].

在进程代数框架下, 并发系统是建立在动作这个基本概念上的, 一个系统 (通常被称为一个进程) 由它所能执行的动作组成，因此，本书假设一个可以观察到的所有动作的集合 Obs，其中的动作用 $a, b, c, \cdots$ 等小写字母来表示 (可能带上下标)，另外, 还有

一个内部不可见的动作 $\tau, \tau \notin$Obs，用于表示系统内部执行的动作. 令 $\mathcal{A} = \text{Obs} \cup \{\tau\}$ 表示所有动作的全集.

定义 3.1.1 设 $a \in \mathcal{A}, A \subseteq \text{Obs}$, Var表示一个进程变量的集合，$x \in$Var. **基本进程代数**BPA 由以下语法产生：

$$P ::= 0 \mid a.P \mid P;P \mid P + P \mid P \parallel_A P \mid P \backslash A \mid x \mid \mu x.P \ .$$

上面定义给出的实际上是进程代数作为一种形式刻画语言的语法构成, 它可以看成是由一个进程的集合及定义在其上的一些操作算子构成的一个代数系统. 先给出这些操作算子的直观含义如下：

- 0 代表中止进程，即不能执行任何动作的进程.
- $a.P$ 是动作前缀操作，表示系统先执行动作 a，然后执行进程 P.
- $P_1;P_2$ 是顺序操作，表示系统先执行进程 P_1 的动作，P_1 成功结束后系统接着执行进程 P_2.
- $P_1 + P_2$ 是选择操作，其含义是系统要么执行进程 P_1，要么执行进程 P_2，两者之间的选择是非确定的，即选择 P_1 或者 P_2 都有可能，具体执行起来是依赖系统运行的环境的. 这种选择的非确定性是并发系统的一个重要特征，也是与传统顺序系统相比的一个重要区别特征.
- $P_1 \parallel_A P_2$ 是并发操作，也是进程代数中最重要的一个操作，它代表了两个进程 P_1 和 P_2 的并发执行, 其中集合 $A \subseteq$Obs 中的动作是两个进程需要同时执行的动作，即需要同步的动作. 如果 $A = \varnothing$，则 P_1 和 P_2 不需要同步任何动作，此时称 P_1 与 P_2 是异步并发执行的，并将 $\parallel_\varnothing$ 简记为 $\parallel\!|$. 另外，为了书写方便，对于只有一个元素的集合 A，将省略表示集合的花括号而直接用该元素来表示该集合.
- $P \backslash A$ 是隐藏操作，或者叫做抽象操作，进程 $P \backslash A$ 的行为与 P 类似，唯一的不同是进程 $P \backslash A$ 中那些在集合 A 中的动作都被看成内部动作 (通过改名为 τ)，即外部将不再可观察到.
- x 和 $\mu x.P$ 是为了支持递归操作而引入的操作符，其中 x 代表一个进程变量，$\mu x.P$ 是递归表达式，其中变量 x 可能在进程 P 中出现形成一个递归函数. 称被递归算子 μ 限制的变量 (如 $\mu x.P$ 中的 x) 为约束 (bound) 变量，没有被递归算子限制的变量称为自由 (free) 变量. 自由变量可以通过替换来实例化 (instantiated)，而约束变量则是不受变量替换的影响，因此约束变量的名字可以被认为是无关紧要的. 一个进程表达式中的某项如果没有自由变量，则称该项为闭项 (closed)，反之称为开项 (open) . 对于递归表达式，要求递归项是良性护卫的 (well-guarded)[50]，这一条件保证了递归项的语义模型与它的指称函数的不动点相对应并且是唯一的. 递归表达式的引入使得进程代数的表达能力更加强大，更适合描述一些特定的系统.

我们假定，这些进程代数的操作算子如果没有用括号来明确表示优先级，则默认按照以下的顺序结合（优先级由高到低排列）：., ;, +, $\|$, $\backslash$.

例 3.1.2 以下是合法的进程代数表达式：

(1)$P_1 = a.b.0 + b.a.0$;

(2)$P_2 = a.(b.0 + c.0)$;

(3)$P_3 = P_1 \|_a P_2$.

进程代数作为一种严格的形式刻画方法，除了要有严格定义的语法结构外，还需要有严格定义的语义模型. 进程代数的形式语义通常用一种称为结构化操作语义（structural operational semantics，SOS）[87] 的方式给出，之所以这么称呼，是因为对于每个语法结构，它递归地定义了表达式行为的操作解释. 结构化操作语义采用下面的表达方式：

$$\frac{\text{前提}1 \wedge \text{前提}2 \wedge \cdots \wedge \text{前提}n}{\text{结论}} \ (\text{条件})$$

称为一条规则，可以读作：如果条件成立的话，该规则可以应用，并且当前提 1 到前提 n 都满足的时候，结论成立. 其中，规则中的前提与结论都具有形式 $P \xrightarrow{a} P'$，表示进程 P 可以执行动作 a，同时演变成进程 P'. 表 3.1 给出了 BPA 的结构化操作语义，其中 $P\{Q/x\}$ 表示将进程 P 中所有进程变量 x 的出现都替换成进程 Q 得到的进程.

表 3.1 BPA的结构化操作语义

$$\frac{}{a.P \xrightarrow{a} P}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1;P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1;P_2 \xrightarrow{a} P_2'}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 + P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 + P_2 \xrightarrow{a} P_2'}$$

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 \|_A P_2 \xrightarrow{a} P_1' \|_A P_2} \ (a \notin A) \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 \|_A P_2 \xrightarrow{a} P_1 \|_A P_2'} \ (a \notin A)$$

$$\frac{P_1 \xrightarrow{a} P_1' \wedge P_2 \xrightarrow{a} P_2'}{P_1 \|_A P_2 \xrightarrow{a} P_1' \|_A P_2'} \ (a \in A)$$

$$\frac{P \xrightarrow{a} P'}{P\backslash A \xrightarrow{a} P'\backslash A} \ (a \notin A) \qquad \frac{P \xrightarrow{a} P'}{P\backslash A \xrightarrow{\tau} P'\backslash A} \ (a \in A)$$

$$\frac{P\{\mu x.P/x\} \xrightarrow{a} P'}{\mu x.P \xrightarrow{a} P'}$$

由该操作语义可以很自然地得到一个转移系统，其中每个转移都形如 $\xrightarrow{a}$，即上面带有一个动作标记，因此被称为标记转移系统（labeled transition systems, LTS）.

LTS 是经典的进程代数的操作语义模型，它是在通常的状态转移图中增加了一个转移动作而得来的. LTS 主要关心的是系统在发生状态转移时所执行的动作，因此是基于动作（action-based）的系统模型. 它的形式定义如下：

定义 3.1.3 一个**标记转移系统** (LTS) 是一个四元组 $(S, \mathrm{Act}, \longrightarrow, s_0)$, 其中

- S 是非空的状态集合;
- $\mathrm{Act} \subseteq \mathcal{A}$ 是动作集合;
- $\longrightarrow: S \times \mathrm{Act} \times S$ 是状态转移关系;
- s_0 是初始状态.

对于 $(s, a, s') \in \longrightarrow$, 通常简记为 $s \xrightarrow{a} s'$. LTS 通常用图形来表示, 如图 3.1 所示. 假定 s_1 是初始状态, 则此时系统既有可能执行动作 a, 也有可能执行动作 b, 因此表示的是系统将在动作 a, b 之间进行选择, 而且这种选择是非确定性的, 有可能依赖于当时系统所处的环境, 当系统选择执行动作 a 后, 只有一个可能的后续动作 b, 反过来, 若系统选择执行动作 b, 则只有一个可能的后续动作 a, 因此, 该标记转移系统表示了这样一个进程的行为: $a.b.0 + b.a.0$. 由此可见, LTS 能给出一个进程代数直观并且精确的执行过程.

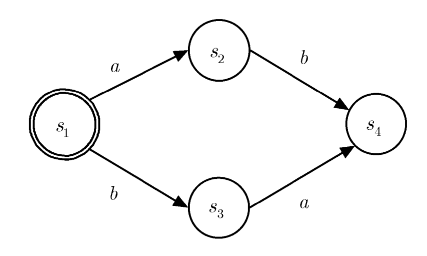


图 3.1 标记转移系统图示

下面考虑两个进程并发执行的操作过程, 先考虑不需要同步的情况. 假设 $P_1 = a.0$, $P_2 = b.0$, 那么 $P_1 \parallel\mid P_2$ 是如何执行的呢? 根据表 3.1 的操作规则, 很容易得出与图 3.1 相同的 LTS 模型, 这表明, 在 LTS 表示的操作语义模型解释下, 可以得出 $a.0 \parallel\mid b.0 = a.b.0 + b.a.0$ 这样的结论, 而这就是前言提到的两个系统并发执行的交织语义理解, 即将两个并发执行的进程看作是两个进程中的动作按照任意可能的交叉顺序执行, 因此说 LTS 是一个交织语义模型. 对于需要同步的情况, 例如, 假设 $P_1 = a.0 + c.0, P_2 = b.0 + c.0$, 则 $P_1 \parallel_c P_2$ 的 LTS 模型如图 3.2 所示.

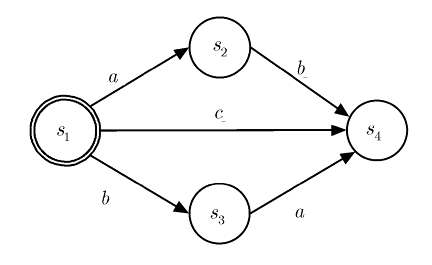


图 3.2 同步并发进程的执行图示

由图 3.2 可以看出，需要同步执行的动作由两个进程同时执行（当两个进程都准备好执行该动作时），而不需要同步的动作则可以按照异步并发的执行方式任意交叉执行.

将上述情况推广到一般情形，设 $P=\sum_i a_i.P_i$，$Q=\sum_j b_j.Q_j$，不难得出，在交织语义下，有下面的展开定律：

$$\begin{aligned}P\parallel_A Q &= \sum_{a_i\notin A} a_i.(P_i\parallel_A Q)+\sum_{b_j\notin A} b_j.(P\parallel_A Q_j)\\ &\quad+\sum_{a_i=b_j\in A} a_i.(P_i\parallel_A Q_j).\end{aligned}\tag{3.1}$$

(3.1) 式给出了两个并发执行的进程严格的交织语义解释，也是进程代数理论中一个重要的等式，如果一个进程代数中允许该式成立，则称为交织语义的进程代数，若不允许该式成立，则该进程代数称为真并发语义的进程代数，如用事件结构 (event structure)[109] 作为语义模型的进程代数 [67].

根据 (3.1) 式及其他进程代数算子的操作语义，可以知道，在交织语义下，任何基本进程代数表达式都可以展开表示为仅包含前缀操作（.）与选择操作（+）的形式，即

$$P=a_1.a_2.\cdots.a_n.0+b_1.b_2.\cdots.b_n.0+\cdots=\sum_{a^i\in Act} a^i_1.a^i_2.\cdots.a^i_n.0\ .$$

因此，在交织语义下，进程代数的推理可以变得相对简单.

例 3.1.4 设 $P=(a.0+b.0)\parallel\!|\,(c.0+d.0)$，则根据展开定律，可以将 P 展开如下：

$$\begin{aligned}P &= (a.0+b.0)\parallel\!|\,(c.0+d.0)\\ &= a.(c.0+d.0)+b.(c.0+d.0)+c.(a.0+b.0)+d.(a.0+b.0)\\ &= a.c.0+a.d.0+b.c.0+b.d.0+c.a.0+c.b.0+d.a.0+d.b.0\ .\end{aligned}$$

3.2 带标记的连续时间马尔可夫链

标记转移系统是一种主要用于刻画并发系统行为特征的模型，它关注的是系统在发生状态转移的时候所执行的动作，状态本身的信息并不是它所关心的，而性能评价领域常用的马尔可夫链模型则主要关心的是系统所处的状态信息，因此，在用马尔可夫链进行系统的性能评价时，人们常常采用稍作修改的马尔可夫链，本节将简要介绍用于性能评价的连续时间马尔可夫链模型.

通常的连续时间马尔可夫链模型就如第二章所介绍的，由转移率矩阵决定它的行为. 但在实际使用当中，除了关心转移率之外，系统所处的状态特征通常也是需要考虑的. 因此，实际中常用的 CTMC 是在系统的状态标记上特定的特征信息，这样的 CTMC 有时也叫做带标记的 CTMC（labeled CTMC）. 与标记转移系统不同的是，这里在状态上标记的信息不是动作，而是一些原子命题，作为该状态满足的一些特征信息. 另外，为了突出状态转移，将转移率矩阵改写为一个转移关系，记为 $\dashrightarrow$. 这样，类似于标记转移系统，可以定义带标记的 CTMC 如下.

假定 AP 是所有原子命题的集合，则

定义 3.2.1 **一个带标记的连续时间马尔可夫链**（labeled CTMC）是一个四元组 $(S, \dashrightarrow, L, \alpha_0)$，其中

- S 是状态空间的集合；
- $\dashrightarrow \subset S \times \mathbb{R}_{\geqslant 0} \times S$ 是状态转移关系，称为马尔可夫转移（Markovian transition）；
- $L : S \mapsto 2^{AP}$ 是标记函数, 用于给每个状态标上该状态所满足的原子命题的集合;
- α_0 是初始分布.

定义中的马尔可夫转移与定义 2.2.2 中的转移率矩阵 $\boldsymbol{R}$ 没有本质的区别，但是却能更直观地表达出转移系统这样一个概念. 与 LTS 类似，对于 $(s, \lambda, s') \in \dashrightarrow$，同样简记为 $s \overset{\lambda}{\dashrightarrow} s'$. 定义中的函数 L 称为标记函数，它是用来给 CTMC 的状态增加一些附加信息的，这些附加信息是一些原子命题的集合. 不失一般性，假设不同的状态满足不同的性质，则这些原子命题的集合就可以用于区分不同的状态，即每个状态都可以由它上面标记的原子命题的集合来唯一标识. 这样一来，带标记的 CTMC 主要关注的就是每个状态上的不同信息了，因此称这样的系统为基于状态的（state-based）系统.

例 3.2.2 图3.3表示一个带标记的CTMC. 其中 $AP = \{a, b, c\}$，每个状态所满足的原子命题标在状态旁，状态间的转移仍然保留虚线箭头的形式. 对于单元素集合，同样采用了省略花括号的简写方式.

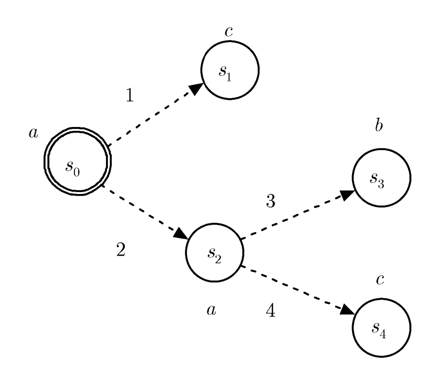


图 3.3 带标记的 CTMC 图示

作为广泛使用的一种性能模型，(带标记的) CTMC 已经得到了很多方面的关注，从它的刻画技术到分析技术，都有许多研究工作并取得了大量研究成果，例如，CTMC 的语言层次的刻画模型有排队网、随机 Petri 网、随机活动网、随机进程代数等，逻辑刻画有连续随机逻辑 CSL 及其变体等，这些研究工作的详细情况请参阅文献 [34, 54, 62, 76, 78, 86]，本书不再一一赘述.

3.3 交互式马尔可夫链 (IMC)

前面两节的内容基本上建立起了两个经典的并发系统的功能模型和性能模型. 本节将提出本书的主要研究对象 —— 交互式马尔可夫链 (IMC)，它实际上是前面两个模型的综合. 由 LTS 和带标记的 CTMC 的定义可以看出，两者是有很多类似的地方的，因此，随着并发系统理论进入更复杂系统的研究，尤其是功能模型和性能模型的统一开始成为研究热点之后，人们自然想到了把这两个模型有机地结合起来.

本节首先简要讨论其他一些结合 LTS 和 CTMC 模型的方法，指出这些方法的不足之处，然后引入交互式马尔可夫链的方法，详细说明它在克服其他方法的不足方面的优越性，这也是我们选择该模型作为研究对象的主要动机之一.

3.3.1 随机进程代数模型

将标记转移系统与连续时间马尔可夫链结合在一起的研究最早可以追溯到文献 [83, 84]. 在这之后，相关的研究工作开始活跃起来，人们提出了各种不同的结合模型[22,55,59~62]，这些模型现在统称为随机进程代数（stochastic process algebra, SPA）. SPA 模型在结合 LTS 和 CTMC 时都有一个共同的特点 —— 它们都是将两个模型不同的转移关系（动作转移与概率转移）融合在一个转移关系中，即将转移 $\xrightarrow{a}$ 和 $\overset{\lambda}{\dashrightarrow}$ 组合成一个单一的转移关系 $\xrightarrow{a,\ \lambda}$，其含义是系统发生该状态转移的延迟时间满足参数为 λ 的指数分布，同时在发生状态转移时系统将执行动作 a. 由于将两种转移关系合二为一，在这种模型中，系统在不同的状态转移之间选择执行的动作将不再是非确定的，因为实际上每个动作的执行都由一个确定的概率分布（由参数 λ 确定的指数分布）来决定，也就是说，非确定性被概率分布所取代. 然而非确定性对于并发系统来说是个非常重要的特征，它有助于模拟很多用其他方式不能表达的重要概念，例如：

- **实现自由** 一个进程的语言描述可以看成是它的一个抽象的设计规范，非确定性在其中就代表了实现的自由，即如果对于某个状态来说有两个转移可以非确定地进行选择，那么在具体实现这个设计规范时就可以在这两个转移之间任选一个而不影响原来的设计规范所表达的功能.

- **调度自由** 这是在交织语义模型下非确定性概念的一个经典的使用，几个并发执行的进程在交织语义理解下可以看成是在这些进程之间自由地选择执行，即为操作系统对并发进程的执行调度提供了较大的自由性和灵活性.

由于 SPA 模型将动作与时间延迟绑定在一个转移关系上，在处理带有同步动作的并行操作时就会遇到一些很棘手的问题. 我们用以下的例子来简单说明一下这个问题.

例 3.3.1 图3.4(a)表示了两个SPA模型，图3.4(b)是它们的同步并发执行在交织语义下得到的模型. 图中进程 P_1 在动作 a 和 c 之间选择执行，同时动作 a 与 b 的执行分别满足参数为 λ 和 ν_1 的指数分布延迟；进程 P_2 在动作 b 和 c 之间选择执行，并且分别满足参数为 μ 和 ν_2 的指数分布延迟；$P_1 \parallel_c P_2$ 表示两个进程并发执行，并且在动作 c 上同步. 图3.4(b)给出了在交织语义下 $P_1 \parallel_c P_2$ 的模型表示，其中，不需要同步的动作 a, b 可以很简单地与原来一样交叉执行，并且交叉执行后每个动作的时间延迟仍然满足同样的指数分布（即参数不变），这是因为指数分布的无记忆性，系统先执行哪个分布延迟对接下来的执行都是没有影响的，所以在图中 $P_1 \parallel_c P_2$ 的模型里，状态 s_{00} 经过参数为 λ 的指数分布延迟执行动作 a 后，在状态 s_{10} 处再选择执行动作 b 时，其满足的延迟分布仍然是参数为 μ 的指数分布.

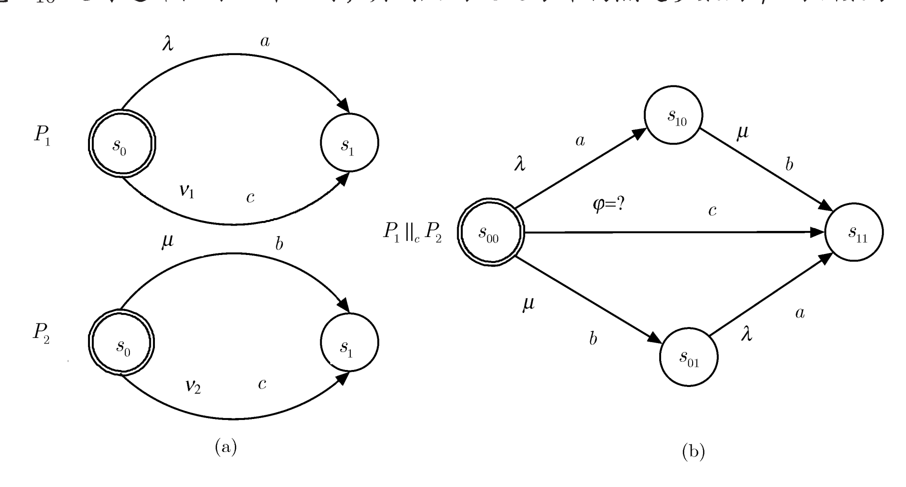


图3.4 SPA模型同步并发执行图示

然而，问题将出在需要同步执行的动作 c 应该满足的概率分布 φ 上. 直观上，同步动作 c 的执行应该在 P_1 与 P_2 都准备好执行 c 的时候，这个时间应该满足以下条件：

(1) 由 ν_1 与 ν_2 决定的指数分布延迟均已完成；

(2) 由 λ 与 μ 决定的指数分布延迟均未完成,

其中，第一个条件由满足参数为 ν_1 与 ν_2 的指数分布的随机变量的最大值决定，

而第二个条件则由满足参数为 λ 与 μ 的指数分布的随机变量的最小值决定，不幸的是，指数分布并不在最大值运算下封闭，即两个满足指数分布的随机变量的最大值不再满足指数分布. 因此，动作 c 的执行所满足的延迟分布不再是指数分布，即图3.4中同步以后动作 c 上的分布参数 φ 无法简单地确定下来（实际上已经跳出了SPA模型的表示范围）. 对这个问题不同的处理方法导致了不同的SPA模型，但是这些解决方法通常都比较复杂，并且直观意义不明确，因此SPA模型总的来说在处理同步并发的时候并不那么令人满意.

总之，随机进程代数模型中有以下两大问题是不能很好解决的：

- 非确定性不再是真正意义上的非确定性，而是被随机分布所代替；
- 带同步的并发操作在 SPA 模型中并不能很好地表示出来，因为指数分布在最大值运算下并不封闭.

交互式马尔可夫链就是为了解决 SPA 这些不足而提出来的另外一种结合 LTS 与 CTMC 的模型. 接下来将具体介绍交互式马尔可夫链模型及其在克服解决上述两个问题的优越性.

3.3.2 交互式马尔可夫链

与 SPA 模型不同的是，交互式马尔可夫链在结合传统的标记转移系统和连续时间马尔可夫链的时候采用的是尽可能正交的方式，即保留两种转移关系，而不是将它们合二为一. 这样，在一个交互式马尔可夫链系统中，就存在两种不同的转移关系. 因此可以给出如下的定义：

定义 3.3.2 一个**交互式马尔可夫链**是一个五元组 $(S, \mathrm{Act}, \longrightarrow, \dashrightarrow, s_0)$，其中

- S 是一个非空的状态集合；
- $\mathrm{Act} \subseteq \mathcal{A}$ 是一个动作集合；
- $\longrightarrow \subseteq S \times \mathrm{Act} \times S$ 是动作转移关系；
- $\dashrightarrow \in S \times \mathbb{R}_{\geqslant 0} \times S$ 是马尔可夫转移关系，满足

$$\forall s_1, s_2 \in S, |(\dashrightarrow \cap (\{s_1\} \times \mathbb{R} \times \{s_2\}))| \leqslant 1;$$

- s_0 是初始状态.

定义中的两个转移关系分别是 LTS 中的动作转移与 CTMC 中的随机分布延迟转移，其含义基本保持不变. 对马尔可夫转移关系要求满足的条件说明，任意一对状态之间至多只有一个马尔可夫转移关系. 如果存在多个这样的转移关系，根据指数分布的性质 3，可以简单地将这几个转移关系合为一个，得到的转移率为原来各个转移率之和. 另外，与转移率矩阵 $\boldsymbol{R}$ 有一点小小的区别是，在马尔可夫转移关系 $\dashrightarrow$ 中，允许到自身的转移，即允许 $s \overset{\lambda}{\dashrightarrow} s$，其中 $\lambda > 0$. 它表示系统经过参数为 λ 的指数分布延迟后仍然停留在原状态（或转移到自身状态）. 这一修改并不影响

IMC 的随机行为，但是却更适合于通常的状态转移解释，如在后面 IMC 的逻辑刻画中所看到的一样.

以下，记 $\boldsymbol{R}(s,s')$ 为状态 s 到 s' 的转移率，即 $(s,\boldsymbol{R}(s,s'),s') \in \dashrightarrow$. s 上的总转移率记为 $E(s) = \sum\limits_{s'\in S} \boldsymbol{R}(s,s')$.

将上述定义与定义 3.1.3 与定义 2.2.2 相比，可以很清楚地看出，IMC 在定义中将 LTS 和 CTMC 的各部分元素基本上是没有改变地结合在一起的. 这样，IMC 就最大可能地保持了 LTS 和 CTMC 两个系统原有的性质. 事实上，这一特点可以用以下的定理来说明.

由于 IMC 定义中的两种转移关系都可以为空集，所以有

定理 3.3.3 [53] *任何一个LTS都与一个IMC同构，任何一个CTMC也与一个IMC同构.*

在 IMC 模型中，没有给每个状态做标记，而主要关心的是系统的转移关系，尤其是动作转移关系，它是能使得 IMC 可以组合化的基础，因此，在这一点上 IMC 继承的是 LTS 的特点，即它也是个基于动作的系统模型，它的目的是提供一个可以组合化的可用于性能评价模型.

根据定义，IMC 的图形表示也可以很方便地结合 LTS 与 CTMC 的图形表示，如图 3.5 所示.

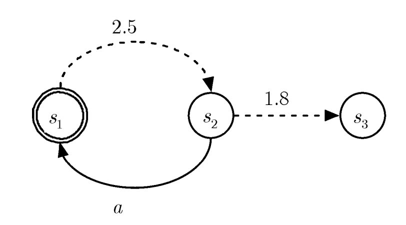


图 3.5　交互式马尔可夫链图示

图 3.5 中，系统从状态 s_1 出发，经过一个参数为 2.5 的指数分布延迟转移到达状态 s_2，在状态 s_2，系统有两个可能的行为，一是执行动作 a 并且回到初始状态，若在状态 s_2 停留一定时间系统仍然未能执行动作 a，则系统进入状态 s_3，表示结束系统的运行. 如图所示，这个决定系统是否结束的时间间隔同样满足参数为 1.8 的指数分布.

在一个 IMC 中，系统在一个状态上可以执行多个可能的动作转移，由于动作转移与时间延迟是分离的，因此这些动作转移之间的选择就可以仍然保留 LTS 的非确定性解释，即系统在不同的动作转移之间的选择是非确定性的，这样，IMC 很容易就解决了 SPA 模型中非确定性不能保留的问题.

若在一个状态上系统既可以执行动作转移，又可以执行马尔可夫转移（如图

3.5 中的状态 s_2)，系统将如何选择呢？IMC 采取的原则是所谓的最大前进假设 (maximal progress assumpltion)：内部动作 (即动作 τ) 的执行不允许时间延迟的发生，即若系统有内部动作要执行的话，该动作的执行是立即的，不允许其他延迟转移发生. 因此，如果一个内部动作转移与马尔可夫转移共同存在与一个状态的时候，马尔可夫转移可以被忽略. 但是, 对于外部动作 (即 Obs 中的动作) a，它的执行是依赖于环境的，或者说它是与环境交互的一个动作，因此是有可能被延迟的，假如在一定时间内系统没有执行动作 a (或与环境进行动作 a 的交互)，那么系统可以选择执行马尔可夫转移. 最大前进假设广泛用于很多实时进程代数中 [7,24,43]，它反映了内部动作具有比外部动作更高的优先级. 图 3.6 说明了在最大前进假设下，图中的两个系统实际上是等价的.

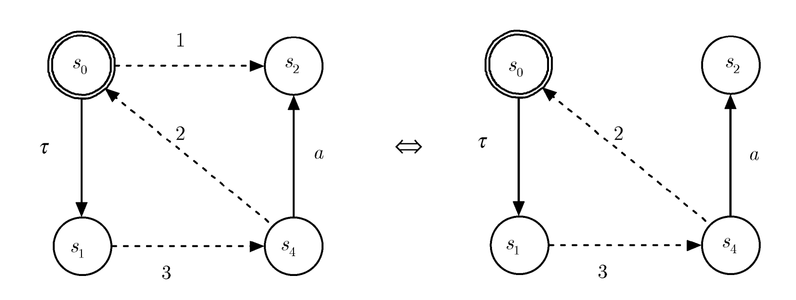


图 3.6 最大前进假设图示

现在来看看 IMC 模型是如何处理两个系统的并发执行的. 在 SPA 模型中，由于动作与延迟结合在同一个转移关系当中，当系统需要同步执行某些动作时，与这些同步动作结合在一起的延迟也不可避免地参与了同步，由此造成了同步动作在同步执行中的延迟满足的随机分布无法简单地确定出来的问题. 而在 IMC 模型中，动作与延迟时间的分离使得我们在考虑同步执行的动作时不必考虑延迟时间，因为它们并不参与同步. 需要同步的动作的执行时间发生在并发执行的所有组件都准备好执行该动作的时候，换句话说，IMC 将同步动作的延迟隐式地实现为最大分布，因为只有动作转移需要同步，因此同步的发生意味着该动作的所有延迟都已经完成. 而对于不需要同步的动作转移以及马尔可夫转移，它们则是按照经典的交织语义来执行的，注意到由于马尔可夫性，马尔可夫转移也是可以任意交叉执行的.

图 3.7 给出了一个简单的同步执行的 IMC 示意图. 假定图中左边两个系统需要同步动作 a，则同步执行的结果就如图中右边系统所示. 这里给出的是直观的一个说明，严格的形式定义将在下一节给出.

由此可见，IMC 很好地克服了 SPA 模型固有的两个不足之处，即它很明确地区分了非确定性和概率，保留了并发系统中非确定性这个重要概念，同时，IMC 在带同步动作的并发执行下，简单而优美地解决了同步动作的随机分布问题. 正是基

于以上这些优点，IMC 成为比 SPA 模型更加适合用于组合性能评价的模型，这也是本书选用这一理论模型的主要动机之一.

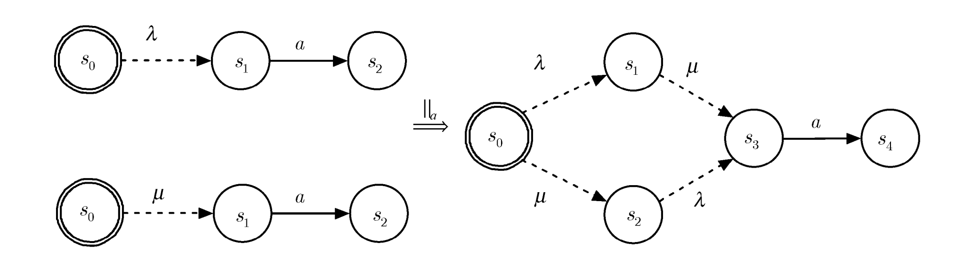


图 3.7　IMC 的同步执行图示

3.4　IMC 的代数刻画

本节将在语言层次给出 IMC 的一种形式刻画. 我们采用的语言是进程代数语言，它实际上是在前面定义的基本进程代数的基础上增加一个所谓的延迟前缀 (delay-prefix) 操作得来的.

定义 3.4.1　令 $a \in \mathrm{Act}, A \subseteq \mathrm{Obs}, \lambda \in \mathbb{R}^+, x \in \mathrm{Var}$，则**交互式马尔可夫链进程代数IMPA** 由以下语法产生:

$$P ::= 0 \mid a.P \mid (\lambda).P \mid P;P \mid P+P \mid P \parallel_A P \mid P \backslash A \mid x \mid \mu x.P\ ,$$

其中，新增加的 $(\lambda).P$ 称为延迟前缀操作，表示系统在执行进程 P 之前需要经过一个参数为 λ 的指数分布时间延迟，它实际上对应了 IMC 模型中马尔可夫转移. 其他的操作算子与前面定义的进程代数语言的含义相同. 可见，IMPA 是 BPA 一个超集.

例 3.4.2　令 $P, Q \in \mathrm{IMPA}$，则以下是合法的 IMPA 表达式：

(1) $a.(\lambda).(\mu).b.(\nu).0$;

(2) $a.P + (\lambda).Q$.

上例中的 (1) 式的直观含义是，系统首先与动作 a 进行交互，执行动作 a，然后，系统在进一步与动作 b 进行交互之前将有两次延迟，第一次满足参数为 λ 的指数分布，第二次满足参数为 μ 的指数分布. 在系统执行动作 b 之后，系统经过一个参数为 ν 的指数分布延迟后中止. (2) 式是 IMPA 中一个典型的表达式，它反映了 IMC 系统中动作转移与马尔可夫转移之间的相互关系. 如上节解释的，(2) 式代表了系统在一个动作转移与马尔可夫转移之间进行选择，这种选择满足最大前进假设，其执行依动作 a 为内部还是外部动作而定. 与 BPA 一样，我们也以结构化操作语义的方式给出 IMPA 严格的形式语义. 由于 IMPA 是在 BPA 的基础上增加了一个延迟前缀得到的，因此 IMPA 的操作语义中就有两种形式的转移，一种是与 BPA

中一样的动作转移 $\xrightarrow{a}$，另一种则是马尔可夫转移 $\stackrel{\lambda}{\dashrightarrow}$，表示系统经过一个参数为 λ 的指数分布延迟转移, 其中动作转移的操作规则仍然满足表 3.1 的规则，相应地，马尔可夫转移的操作规则由表 3.2 给出.

表 3.2 IMPA马尔可夫转移的结构化操作语义

$$\frac{}{(\lambda).P \stackrel{\lambda}{\dashrightarrow} P}$$

$$\frac{P_1 \stackrel{\lambda}{\dashrightarrow} P_1'}{P_1; P_2 \stackrel{\lambda}{\dashrightarrow} P_1'} \qquad \frac{P_2 \stackrel{\lambda}{\dashrightarrow} P_2'}{P_1; P_2 \stackrel{\lambda}{\dashrightarrow} P_2'}$$

$$\frac{P_1 \stackrel{\lambda}{\dashrightarrow} P_1' \wedge P_2 \stackrel{\tau}{\not\longrightarrow}}{P_1 + P_2 \stackrel{\lambda}{\dashrightarrow} P_1'} \qquad \frac{P_2 \stackrel{\lambda}{\dashrightarrow} P_2' \wedge P_1 \stackrel{\tau}{\not\longrightarrow}}{P_1 + P_2 \stackrel{\lambda}{\dashrightarrow} P_2'}$$

$$\frac{P_1 \stackrel{\lambda}{\dashrightarrow} P_1' \wedge P_2 \stackrel{\tau}{\not\longrightarrow}}{P_1 \parallel_A P_2 \stackrel{\lambda}{\dashrightarrow} P_1' \parallel_A P_2} \qquad \frac{P_2 \stackrel{\lambda}{\dashrightarrow} P_2' \wedge P_1 \stackrel{\tau}{\not\longrightarrow}}{P_1 \parallel_A P_2 \stackrel{\lambda}{\dashrightarrow} P_1 \parallel_A P_2'}$$

$$\frac{P \stackrel{\lambda}{\dashrightarrow} P'}{P \backslash A \stackrel{\lambda}{\dashrightarrow} P' \backslash A}$$

$$\frac{P\{\mu x.P/x\} \stackrel{\lambda}{\dashrightarrow} P'}{\mu x.P \stackrel{\lambda}{\dashrightarrow} P'}$$

表中 $P \stackrel{\tau}{\not\longrightarrow}$ 表示进程 P 不能以内部动作 τ 开始执行. 注意在 P_1+P_2 与 $P_1 \parallel_A P_2$ 的操作规则中，除了要求其中一个操作能执行马尔可夫转移之外，同时还要求另外一个操作不能执行内部动作转移, 这一额外要求就体现了前面所说的最大前进假设，即若系统能在两个子进程中进行选择的话，并且要能选择执行马尔可夫转移，则不能同时存在内部动作的执行，否则，系统将优先执行内部动作而忽略掉可能的马尔可夫转移.

另外，两个并发执行的进程，由于同步动作集合与延迟分布没有关系，因此不存在对马尔可夫转移的同步, 此时两个并发执行的进程中的马尔可夫转移可以按照任意的交叉顺序来执行，而真正的同步由动作转移满足的规则来决定（如表 3.1）.

根据上面定义的 IMPA 并发算子的操作语义, 可以得出 IMPA 在交织语义下的展开定律，设

$$P = \sum_i a_i.P_i + \sum_j (\lambda_j).P_j, \quad Q = \sum_k b_k.Q_k + \sum_l (\mu_l).Q_l,$$

则

$$\begin{aligned} P \parallel_A Q \quad = \quad & \sum_{a_i \notin A} a_i.(P_i \parallel_A Q) + \sum (\lambda_j).(P_j \parallel_A Q) \\ & + \sum_{b_k \notin A} b_k.(P \parallel_A Q_k) + \sum (\mu_l).(P \parallel_A Q_l) \\ & + \sum_{a_i = b_k \in A} a_i.(P_i \parallel_A Q_k). \end{aligned} \tag{3.2}$$

与 (3.1) 式比较一下很容易得出,(3.2) 式实际上是 (3.1) 式的一个直接扩展, 增加的随机分布延迟并没有影响原来的动作转移操作, 因此说这种扩展是一种正交的扩展, 它反映了 IMC 在组合传统的 LTS 与 CTMC 模型的时候最大可能地保持了两个模型原有的性质, 并因此得到了良好的性质.

根据 (3.2) 式及 IMPA 的其他算子的操作语义, 同样可以将一个 IMPA 表达式展开表示成以下只含前缀操作 (.) 和选择操作 (+) 的标准形式:

$$P = \sum_{\gamma^i \in \mathrm{Act} \cup (\mathbb{R}^+)} \gamma_1^i . \gamma_2^i . \cdots . \gamma_n^i . 0, \tag{3.3}$$

其中, $(\mathbb{R}^+) = \{(\lambda) \mid \lambda \in \mathbb{R}^+\}$. 这样, 基于 IMPA 的推理也可以大大简化了.

例 3.4.3 图3.8显示了根据IMPA的操作语义 (表3.2及表3.1) 得出的两个并发执行的IMPA进程的操作模型.

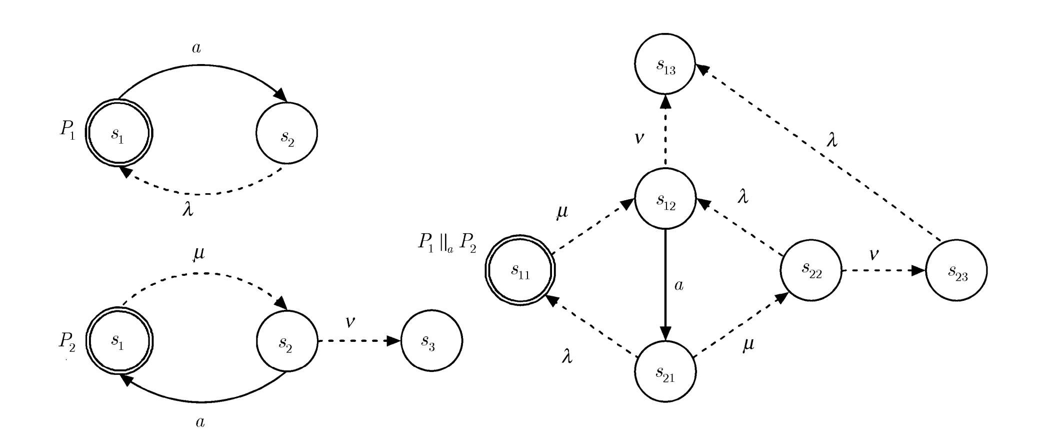


图 3.8 两个 IMPA 进程的同步图示

由此可见, IMPA 的结构化操作语义在 IMPA 与 IMC 之间建立了一个对应关系, 如同 BPA 的操作语义在 BPA 与 LTS 之间建立了一个对应关系一样. 这样, 给定一个 IMPA 表达式, 就能得到一个 IMC 模型, 它给出了这个表达式的精确语义 (注意, 反过来不一定成立, 即给定一个 IMC 模型, 有可能对应多个表达式). 从这个意义上来说, 可以把 IMPA 看作是 IMC 的一个形式刻画语言, 而 IMC 则是 IMPA 的形式语义模型.

建立 IMC 的代数刻画的目的是在更抽象的语言层次上对 IMC 进行研究, 因为形式语言只涉及语法和公理系统, 所以更易于使用. 本节只建立了 IMPA 的语法和语义框架, 要使用 IMPA 对并发系统进行推理分析还需要建立其上的公理系统, 这是与等价关系的概念紧密结合在一起的, 我们将在第四章进行详细讨论.

3.5 IMC 的逻辑刻画

本节将从逻辑角度对 IMC 模型进行刻画. 如第一章所述, 并发系统的研究可以在三个层次上进行, 即语言、结构和逻辑. 并发系统研究所采用的逻辑通常是时序逻辑, 这是一种用于刻画随时间变化的系统性质的逻辑, 它是在经典的命题逻辑中增加一些用于描述时序 (如 X (下一步)、$\mathcal{U}$ (直到)、F (将来)、G (总是) 等) 特征的算子 (运算) 而得到的.

时序逻辑按照其描述的时序特征是线性时间还是分支时间可以分为两大类, 一类是线性时间时序逻辑, 如 LTL, 一类是分支时间时序逻辑, 如 CTL. 时序逻辑由于模型检验技术的成功而广泛用于并发系统的形式验证领域, 通过时序逻辑, 可以获得很强的表达能力来刻画一个系统所满足的一些关键性质, 如活性、安全性等.

时序逻辑最早用于刻画系统的功能方面的特性, 随着研究的不断深入, 用于描述实时系统或随机系统特征的逻辑也被建立起来, 如 RTCTL, PCTL,CSL 等, 这些逻辑基本上都是在 CTL 的基础上进行一定的扩展, 增加一些用于描述数量或随机特性的算子得到的, 这些扩展使得时序逻辑可以进一步表达系统性能方面的特征, 成为基于模型检验的性能分析方法的重要基础.

由于经典的时序逻辑都是基于状态的, 即它们都假设系统在每个状态上有特定的性质, 根据这些特定的性质来刻画系统的时序特性. 如一个典型的 CTL 公式为 $\mathbf{AG}p$, 表示系统在今后所有执行路径上的状态都满足性质 p. 而前面已经说过, IMC 是一种基于动作的模型, 它不关心系统状态上的信息, 而主要关心系统所能执行的动作, 因此那些基于状态的时序逻辑无法对 IMC 进行刻画. 本书针对 IMC 模型的特点定义了一种新的逻辑, 与其他逻辑不同的是, 该逻辑是基于动作的时序逻辑, 命名为 aCSL (action-based CSL), 它实际上是 CSL 逻辑的一种变体. 本节将详细定义 aCSL 的语法和语义, 从而建立起 IMC 在逻辑层次上的形式刻画.

3.5.1 IMC 的路径及其上的概率

由于时序逻辑通常用于描述系统随时间变化的时序特征, 因此常常跟系统的执行路径相关, 首先需要定义 IMC 上的路径概念. 当一个 IMC 描述的系统运行时, 系统从一个状态转移到另一个状态, 有两个特征是需要表示出来的, 一个是系统在前一个状态的停留时间, 另一个就是系统可能执行 (或与外界交互) 的动作. 为了将这两个特征用一个统一的方式表达出来, 我们定义一个二元组 (a,t) 来表示这两个特征, 其中第一个分量表示系统在状态转移中可能执行的动作, 第二个分量表示系统在执行状态转移前所延迟的时间. 如果发生的转移是马尔可夫转移, 即不需要与任何动作进行交互, 则表示为 $(*,t)$.

定义 3.5.1 设 $M = (S, \mathrm{Act}, \longrightarrow, \dashrightarrow, s_0)$ 是一个IMC, 令 $\mathrm{Label} = (\mathrm{Act} \cup$

$\{*\}) \times \mathbb{R}_{\geqslant 0}$，一条IMC上的**路径**是以下一个序列：

$$\sigma = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \cdots s_{k-1} \xrightarrow{l_{k-1}} s_k, \quad s_i \in S, l_i = (a_i, t_i) \in \text{Label}, i \in \mathbb{N},$$

满足从状态 s_i 到 s_{i+1} 存在一个动作转移或马尔可夫转移.

如果上述定义中 s_k 不再可以执行任何转移, 称这条路径是有限的, 否则它就是无限的. 用 $\text{Path}(s)$ 来表示从 s 出发的所有路径的全体集合.

为了叙述方便, 再定义以下记号. 对于一条无限路径 σ, 用 $\sigma[i]$ 表示它的第 $i+1$ 个状态，即 $\sigma[i] = s_i$，在状态 s_i 上的停留时间用 $\delta(\sigma, i)$ 表示，即 $\delta(\sigma, i) = t_i$. 对于一个给定的时刻 t, σ 在该时刻所处的状态用 $\sigma@t$ 表示, 则 $\sigma@t = \sigma[i]$, 其中 i 是满足 $t \leqslant \sum_{j=0}^{i} \delta(\sigma, j)$ 的最小的 i. 对于一条结束于 s_k 的有限路径 σ，$\sigma[i]$ 和 $\delta(\sigma, i)$ 只对 $i \leqslant k$ 有定义. 最后，对于 $L \subseteq \text{Label}$，用 $\sigma[i] \xrightarrow{L} \sigma[i+1]$ 表示 $\sigma[i]$ 能通过一个由 L 中的元素刻画的转移到达状态 $\sigma[i+1]$.

为了刻画 IMC 的性能特征, 需要在 IMC 的路径上定义概率测度, 令 $C(s_0, I_0, \cdots, I_{k-1}, s_k) = \{\sigma \in \text{Path}(s_0) \mid \sigma[i] = s_i (i \leqslant k), \delta(\sigma, i) \in I_i \subseteq \mathbb{R}_{\geqslant 0} (i < k)\}$，表示从 s_0 出发的所有满足每一步转移的延迟时间在相应的区间范围内的路径的集合. 则 IMC 路径上的概率 Pr 可以归纳定义如下：

(1) $\Pr(C(s_0)) = 1$,

(2) For $k \geqslant 0, a \in \text{Act}$,

$$\Pr(C(s_0, I_0, \cdots, I_{k-1}, s_k, I', s'))$$
$$= \begin{cases} \Pr(C(s_0, I_0, \cdots, s_k)) \cdot \int_{I'} \boldsymbol{R}(s_k, s') \cdot e^{-E(s_k)\cdot t} dt, & \text{若} s_k \xrightarrow{(*, t')} s', \\ \Pr(C(s_0, I_0, \cdots, s_k)), & \text{若} s_k \xrightarrow{(a, t')} s'. \end{cases}$$

需要说明的是, 由于 IMC 中既存在概率转移, 又存在非确定性动作转移, 因此, 需要先将 IMC 中的非确定性动作转移以某种方式确定下来, 而 IMC 路径上的概率则是定义在将非确定性动作转移确定下来之后的模型上的.

将非确定动作转移确定下来的方法是由外部给定每个动作转移发生的时间, 这个时间作为一个确定时间, 需要从路径概率的计算中去除. 如果从一个状态出发同时存在多个动作转移, 那么实际发生的动作转移以给定的发生时间中最小的为准, 这样, 非确定性的动作转移就可以确定下来, 得到的 IMC 就只含有随机 (指数分布) 与确定 (由外部环境给定) 的时间指标. 因此在 IMC 路径上的概率的定义中, 遇到动作转移时, 转移关系上标明的时间是一个确定的时间, 而该动作转移的发生也是确定的, 与系统的随机行为无关, 因此在 IMC 路径上的概率的定义当中, 动作转移被认为是以概率 1 通过, 并且花费了确定的时间 t.

注意到在上面的计算中，

$$\int_{I'} \boldsymbol{R}(s_k, s') \cdot e^{-E(s_k)\cdot t} dt = \int_{I'} \frac{\boldsymbol{R}(s_k, s')}{E(s_k)} \cdot E(s_k) \cdot e^{-E(s_k)\cdot t} dt,$$

其中 $E(s_k) \cdot e^{-E(s_k)\cdot t}$ 是系统在状态 s_k 上停留 t 个时间单位的概率密度，而 $\dfrac{\boldsymbol{R}(s_k, s')}{E(s_k)}$ 则是系统从状态 s_k 转移到 s' 的概率，因此 $\int_{I'} \boldsymbol{R}(s_k, s') \cdot e^{-E(s_k)\cdot t} dt$ 实际上表示了系统在时间 $t \in I'$ 内从状态 s_k 转移到状态 s' 的概率.

由于动作转移与时间延迟是分离的，因此在上面的定义中，可以简单地忽略动作转移，它对整个路径的概率计算并不造成影响.

3.5.2 aCSL 逻辑的语法

现在，可以给出 aCSL 逻辑的语法定义了. 它实际上是在 CTL 逻辑的随机化版本 CSL 的基础上修改得来的，在 aCSL 中，去掉了传统的原子命题，增加了动作的描述，因此 aCSL 是个基于动作的时序逻辑系统.

定义 3.5.2 设 $p \in [0, 1], \bowtie \in \{\leqslant, <, \geqslant, >\}$，aCSL的**状态公式**由以下语法产生：

$$\Phi ::= \text{true} \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}(\varphi),$$

对 $t \in \mathbb{R}_{>0}, A, B \subseteq \text{Act}$，aCSL的**路径公式**由以下语法产生：

$$\varphi ::= \Phi_A \mathcal{U}^{<t} \Phi \mid \Phi_A \mathcal{U}^{<t}{}_B \Phi.$$

状态公式表示系统的状态应该满足的性质，路径公式表示系统的执行路径应该满足的性质. 与传统的 LTL,CTL 等逻辑不同的是，在 aCSL 逻辑的状态公式当中，没有出现原子命题，这表明 aCSL 逻辑不再是基于状态的逻辑了. 由状态公式中的 $\wedge, \neg$ 可以推出其他所有的命题逻辑连接词. 状态公式中最主要的就是 $\mathcal{P}_{\bowtie p}(\varphi)$，这个公式的含义是满足路径公式 φ 的路径集合的概率测度在 $\bowtie p$ 所限定的范围内. 这个算子实际上代替了通常的 CTL 路径量词 $\exists$ 和 $\forall$，直观上，$\exists\varphi$ 在 aCSL 中对应于 $\mathcal{P}_{>0}(\varphi)$，而 $\forall\varphi$ 则对应于 aCSL 公式 $\mathcal{P}_{\geqslant 1}(\varphi)$.

路径公式中的算子 $\mathcal{U}^{<t}$ 是 CTL 中 $\mathcal{U}$ 算子的一个带时间的变体，表示完成这样一条路径的时间必须满足在时间 t 之内. 这里，为了简化公式的复杂性，只考虑 "$< t$" 形式的时间界，其他形式的时间界可以通过映射到该区间来得到调整.

下面具体解释一下路径公式的含义. 为了叙述方便，称满足状态公式 Φ 的状态为 Φ 状态，一个执行集合 A 中的动作的转移称为 A 转移. 路径公式 $\Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2$ 的直观含义是一条最终到达 Φ_2 状态的路径，并且在到达 Φ_2 状态之前，这条路径所经过的状态都满足状态公式 Φ_1（即只经过 Φ_1 状态），同时状态的转移只能执行集合 A 中的动作，此外，从这条路径开始到到达 Φ_2 状态所经过的时间至多不超过 t. 路

径公式 $\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$ 的意义与 $\Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2$ 类似，唯一的不同是该路径要求最后到达 Φ_2 状态的转移必须执行，且执行的动作在集合 B 中. 对此可以作如下说明. 由于在公式 $\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$ 中，Φ_2 状态必须要从 Φ_1 状态通过一个 B 转移到达，所以该公式在一个 $(\neg\Phi_1 \wedge \Phi_2)$ 状态 s 是无效的，因为该状态尽管满足 Φ_2，但由于不满足 Φ_1，所以不存在从 Φ_1 状态到 Φ_2 状态的转移. 然而，对于公式 $\Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2$ 来说，它在上述状态 s 却是有效的，因为该公式不要求 Φ_2 状态必须经过一个转移到达，只要当前状态满足 Φ_2，该公式就是成立的. 这一点区别将在后面给出 aCSL 公式的语义时得到严格的定义说明.

aCSL 中的路径公式与“标准”的线性时间与分支时间时序逻辑中的 $\mathcal{U}$ 公式 $\Phi_1\mathcal{U}\Phi_2$ 的区别是：(1) 状态转移需要限制所执行的动作；(2) 到达 Φ_2 状态所需要的时间也有限制. 它们之间的关系可以用下面的式子得到更精确地说明：

$$\Phi_1\mathcal{U}\Phi_2 = \Phi_1\ {}_{\mathrm{Act}}\mathcal{U}^{<\infty}\Phi_2.$$

在通常的时序逻辑当中，还有一个很重要的路径公式 $X\Phi$，表示该路径下一步的状态满足 Φ. 有趣的是，在 aCSL 逻辑当中，这个算子可以用 $\mathcal{U}$ 算子推导出来，即

$$\begin{aligned} X_A^{<t}\Phi &= \text{true}\ {}_{\varnothing}\mathcal{U}^{<t}{}_A\Phi, \\ X_A\Phi &= X_A^{<\infty}\Phi, \\ X\Phi &= X_{\mathrm{Act}}\Phi. \end{aligned}$$

公式 $X_A^{<t}\Phi$ 断言从当前状态出发，在时间 t 内可以通过一个 A 转移到达 Φ 状态. 注意在这里 Φ 状态必须是在第一个动作转移下达到，并且不能有更进一步的动作转移. 由此可见，在 aCSL 中，下一步的概念是建立在动作转移的基础上的，而并非每次状态的改变.

最后，来看看其他常用的时序逻辑算子在 aCSL 中是如何表示的. 首先是 F 算子 (又写作 $\diamond$)，Fp 表示从当前状态出发，将来某个时候一定会达到一个 p 状态. 在 aCSL 中，相应的公式可以表示为

$$\begin{aligned} {}_AF^{<t}\Phi &= \text{true}\ {}_A\mathcal{U}^{<t}\Phi, \\ {}_AF\Phi &= {}_AF^{<\infty}\Phi, \\ F\Phi &= {}_{\mathrm{Act}}F\Phi. \end{aligned}$$

一条路径满足 ${}_AF^{<t}\Phi$ 当且仅当它在时间 t 内到只通过 A 转移到达一个 Φ 状态. 另外，还可以定义以下更丰富的 F 算子：

$$\begin{aligned} {}_AF_B^{<t}\Phi &= \text{true}\ {}_A\mathcal{U}^{<t}{}_B\Phi, \\ {}_AF_B\Phi &= {}_AF_B^{<\infty}\Phi. \end{aligned}$$

在通常的时序逻辑中，G 算子与 F 算子是对偶的一个算子，Gp 表示 p 在将来所有状态都成立. 在 aCSL 中，也有以下的关系：

$$\begin{aligned}\mathcal{P}_{\bowtie p}({}_AG^{<t}\Phi) &= \neg\mathcal{P}_{\bowtie p}({}_AF^{<t}\neg\Phi),\\ \mathcal{P}_{\bowtie p}({}_AG_B^{<t}\Phi) &= \neg\mathcal{P}_{\bowtie p}({}_AF_B^{<t}\neg\Phi).\end{aligned}$$

3.5.3 aCSL 逻辑的语义

前面给出了 aCSL 的语法定义和直观的解释，本节将给出 aCSL 严格的语义定义. 实际上，它的语义模型就是 IMC，准确地说，aCSL 的状态公式是在 IMC 的状态上解释的，而 aCSL 的路径公式则是在 IMC 的路径上解释的.

令 $M = (S, \mathrm{Act}, \longrightarrow, \dashrightarrow, s_0)$ 是一个 IMC，则 aCSL 状态公式和路径公式的语义由以下定义在 IMC 状态和路径上的满足关系 $\models$ 给出.

定义 3.5.3 令 $\mathrm{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$，aCSL状态公式的满足关系 $\models$ 定义如下：

$$\begin{array}{ll} s \models \mathrm{true}, & \text{对任意的} s \in S,\\ s \models \neg\Phi, & \text{当且仅当} s \not\models \Phi,\\ s \models \Phi_1 \wedge \Phi_2, & \text{当且仅当} s \models \Phi_1 \wedge s \models \Phi_2,\\ s \models \mathcal{P}_{\bowtie p}(\varphi), & \text{当且仅当}\mathrm{Prob}(s,\varphi) \bowtie p. \end{array}$$

这里，$\mathrm{Prob}(s,\varphi)$ 表示 $\mathrm{Path}(s)$ 中所有满足路径公式 φ 的路径的概率测度，即

$$\mathrm{Prob}(s,\varphi) = \mathrm{Pr}\{\sigma \in \mathrm{Path}(s) \mid \sigma \models \varphi\}.$$

定义 3.5.4 令 $L_A^* = (A \cup \{*\}) \times \mathbb{R}_{\geqslant 0}, L_A = A \times \mathbb{R}_{\geqslant 0}, L_B = B \times \mathbb{R}_{\geqslant 0}$，aCSL路径公式的满足关系 $\models$ 定义如下：

$$\begin{array}{ll} \sigma \models \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2\ \text{当且仅当}\exists k \geqslant 0. & (\sigma[k] \models \Phi_2\\ & \wedge(\forall 0 \leqslant i < k-1. \sigma[i] \models \Phi_1 \wedge \sigma[i] \xrightarrow{L_A^*} \sigma[i+1])\\ & \wedge \sigma[k-1] \models \Phi_1 \wedge \sigma[k-1] \xrightarrow{L_A} \sigma[k]\\ & \wedge \sum_{i=0}^{k-1}\delta(\sigma,i) < t);\\ \sigma \models \Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2\ \text{当且仅当}\exists k > 0. & (\sigma[k] \models \Phi_2\\ & \wedge(\forall 0 \leqslant i < k-1. \sigma[i] \models \Phi_1 \wedge \sigma[i] \xrightarrow{L_A^*} \sigma[i+1])\\ & \wedge \sigma[k-1] \models \Phi_1 \wedge \sigma[k-1] \xrightarrow{L_B} \sigma[k]\\ & \wedge \sum_{i=0}^{k-1}\delta(\sigma,i) < t). \end{array}$$

注意到 $\delta(\sigma,i)$ 表示系统在状态 $\sigma[i]$ 的停留时间. 由以上定义很清楚地可以看出，公式 $\Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2$ 对于一条从 Φ_2 状态出发的路径总是满足的，不管这条路径将来是怎么执行的，因为 $\sigma[0] \models \Phi_2$. 而对于公式 $\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$，必须至少考虑一次动作转移，即最后一次动作转移必须执行 B 转移.

例 3.5.5 给定一个IMC如图3.9所示，设有路径公式

$$\varphi_1 = \text{true } {}_a\mathcal{U}^{<10}\text{true},$$

$$\varphi_2 = \text{true } {}_a\mathcal{U}^{<10}{}_b\text{true},$$

则 φ_1 刻画了一条在10个时间单位内从 s_0 出发，最后停留在 s_0, s_1, s_3 中的一个状态的路径，而 φ_2 则刻画了一条在10个时间单位内从 s_0 出发，最后必须停留在 s_2 状态的路径.

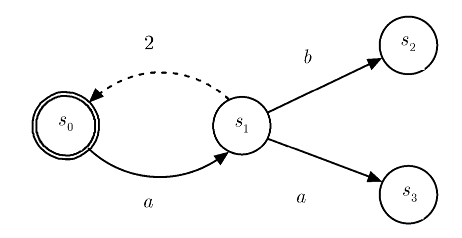


图 3.9 aCSL 路径公式的例子

这样就建立起了一个 IMC 的逻辑刻画系统 aCSL，它能对 IMC 的动态行为作出严格的刻画，具有很强的表达能力，同时也为第六章对 IMC 进行模型检验的研究奠定了良好的逻辑基础.

第四章　分支时间等价和前序关系

4.1　概　　述

在并发系统理论当中，等价关系一直是个核心的研究内容. 在传统的功能分析领域，等价关系能够建立不同系统之间行为比较的标准，同时也是进程代数理论中各种代数公理建立的基础. 而在性能评价领域，人们对广泛使用的马尔可夫链模型也定义了相关的等价关系，用来对状态空间进行聚集，达到压缩状态空间的目的.

传统的等价关系是定义在功能模型上的，如标记转移系统、事件结构等. 在这些模型上，人们定义了从线性时间到分支时间的各种等价关系，其中最值得一提的是 Rob J. van Glabbeek 的工作，他在传统的并发系统功能模型上将各种常用等价关系详细分类，提出了从迹等价到互模拟等价共计 11 种等价关系，并且揭示了这些等价关系的性质及其联系，其研究结果表明，互模拟等价具有良好的性质，并且具有比其他等价关系更强的区分能力.

由于分支时间等价关系在传统的功能模型上得到了广泛的应用，人们很自然地想到将其思想应用到概率系统上. 这方面的工作最初由文献 [65, 69] 开始，此后，各种概率和随机模型上的（互）模拟等价的概念都被建立起来 [5,14,16,21,25,62,66,85,93,105]，成为对随机系统进行推理分析的重要工具.

在这些等价关系的研究当中，除了定义和性质是一方面的研究重点外，与这些等价关系相关的逻辑特征也得到了较为广泛的研究，例如经典的功能模型上的互模拟等价关系可以用 Hennesy-Milner 逻辑进行描述 [52]，而基于 CTL 时序逻辑的互模拟等价关系的性质研究也获得了非常重要的结果 [26]. 在性能模型方面，基于 DTMC 和 CTMC 的逻辑 PCTL 和 CSL 与 DTMC 和 CTMC 上的互模拟等价关系之间的联系也得到了深入研究 [1,14,17,18,38,39,93].

然而，所有这些已经取得的结果都是相对独立的，它们之间的联系很少得到关注，本章将以 IMC 作为载体，研究这些已有的分支时间等价关系之间的联系，包括强（互）模拟和弱（互）模拟等价关系. 由于 IMC 结合了传统的功能模型与性能模型，因此定义的这些分支时间等价关系必须既能反映出功能特征，又能反映出性能特征，这一点通过与 IMC 类似的正交结合功能模型与性能模型上的分支时间等价关系得到了保证. 同时，本章对于定义的分支时间等价关系的逻辑特征也进行了探讨，从而使得 IMC 上定义的等价关系更加系统和完善，本章的研究结果基本上涵盖了已有的分支时间等价关系的研究结果.

4.2 互模拟等价关系

直观上，互模拟等价关系反映的是两个系统的行为可以相互模拟. 这在传统的功能模型上比较容易理解，只需要考虑两个系统分别能执行什么动作以及怎么执行的，对于以随机行为表示的性能模型，则需要定义什么是相互模拟的随机行为. 由于 IMC 的随机行为由 CTMC 刻画，因此，随机系统上的相互模拟概念就变成了找到 CTMC 上的一个等价类划分所满足的条件的问题.

本节将在 IMC 模型上给出互模拟等价关系所满足的条件，我们遵循经典的等价关系的划分，分别给出强互模拟和弱互模拟的定义，并讨论它们之间的相互联系.

以下，如果没有特别说明，假设所有的关系都定义在交互式马尔可夫链 $M = (S, \mathrm{Act}, \longrightarrow, \dashrightarrow, s_0)$ 上.

4.2.1 强互模拟等价

定义 4.2.1 一个定义在 S 上的等价关系 $\mathcal{B}$ 称为一个**强互模拟等价关系**，当且仅当对所有的 $s_1\mathcal{B}s_2$，以下条件成立：

(1) $s_1 \overset{a}{\longrightarrow} s_1', a \in \mathrm{Act} \;\Rightarrow\; \exists s_2' \in S, s_2 \overset{a}{\longrightarrow} s_2'$ 并且 $s_1'\mathcal{B}s_2'$;

(2) 对所有的 $\mathcal{B}$ 等价类 C，有 $s_1 \overset{\tau}{\not\longrightarrow} \Rightarrow \boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C)$.

如果存在一个 S 上的强互模拟等价关系 $\mathcal{B}$，使得 $s_1\mathcal{B}s_2$，则 s_1 与 s_2 称为强互模拟的，记为 $s_1 \sim s_2$.

定义中 $\boldsymbol{R}(s, C) = \sum_{s' \in C} \boldsymbol{R}(s, s')$，表示从状态 s 出发经过一个马尔可夫转移到 C 中所有状态的转移率的累加和. 第一个条件实际上反映的是传统的功能模型上的互模拟等价要求，即动作转移需要满足的条件，而第二个条件则是随机模型上的互模拟等价要求，即马尔可夫转移需要满足的条件，它反映了在 CTMC 模型下，一个互模拟等价关系类的划分必须满足同一个等价类中的状态到所有等价类的状态转移率之和都相等，这里，根据交互式马尔可夫链中两种转移关系之间所满足的最大前进假设，增加了限制条件 $s_1 \overset{\tau}{\not\longrightarrow}$，即只有在 s_1 不能执行内部动作转移的条件下，s_1 上的转移率才需要考虑.

图4.1是一个给定的IMC上的强互模拟等价关系的例子. 图中相同阴影填充的状态是强互模拟的状态. 定义 4.2.1 中的两个条件可以很容易在图中得到检验. 例如，图中填充为（⦸）的状态是强互模拟的状态，它们都能执行一个 a 动作转移到另一个等价状态（⦶）（注意，由于等价关系的自反性，标记为（⦶）的状态自身也构成一个等价类），同时，它们到每一个等价类（包括自身）的转移率的和都是相等的，如对（⦸）中的每一个状态 s，都有

$$\boldsymbol{R}(s, ⦸) = 0, \quad \boldsymbol{R}(s, ⊘) = 5, \quad \boldsymbol{R}(s, ⦶) = 0.$$

对于另外两个等价类，也可以类似验证.

将图4.1中IMC的状态按强互模拟等价关系划分，则图4.1可以简化为图4.2，其中相互等价的状态用一个状态来代表，这样实际上得到了一个压缩的IMC. 将在第六章看到这种状态空间压缩在模型检验中的应用.

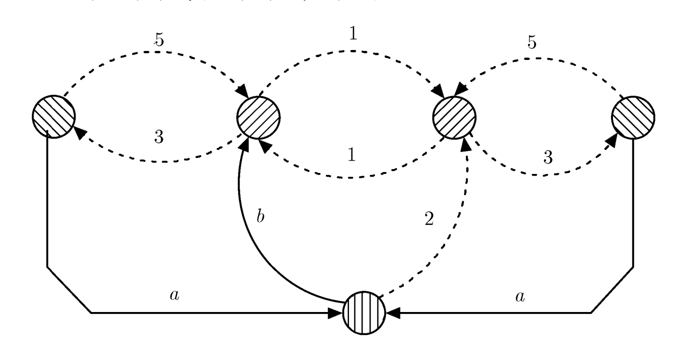


图4.1 强互模拟等价关系图例

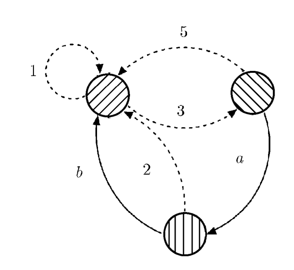


图4.2 强互模拟等价关系状态空间压缩图示

4.2.2 弱互模拟等价

上面定义的强互模拟等价关系中，内部动作只在马尔可夫转移中作了特殊对待以满足最大前进假设，在动作转移所满足的条件中，内部动作与外部动作是同等对待的. 然而，在经典的 LTS 上定义的各种等价关系中，有时候不需要关心内部动作的执行情况，即只考虑外部可见动作的执行所满足的条件，这样定义的关系通常称为弱关系. 遵循这样的思想，忽略掉被认为是内部执行的行为，就可以得到 IMC 上的弱互模拟等价关系的概念.

令 $\stackrel{\tau}{\Longrightarrow}$ 表示 $\stackrel{\tau}{\longrightarrow}$ 的自反传递闭包，$\stackrel{a}{\Longrightarrow}$ 表示 $\stackrel{\tau}{\Longrightarrow}\stackrel{a}{\longrightarrow}\stackrel{\tau}{\Longrightarrow}$. 注意，$\stackrel{\tau}{\Longrightarrow}$ 有可能没有执行任何的内部动作转移（空操作），而 $\stackrel{a}{\Longrightarrow}$ 则必须执行一个且只能执行一个 a 动作转移，之前和之后可以跟着任意数目（包括 0）的内部动作转移.

定义 4.2.3 一个定义在 S 上的等价关系 $\widetilde{\mathcal{B}}$ 称为一个**弱互模拟等价关系**，当且仅当对所有的 $s_1\widetilde{\mathcal{B}}s_2$，以下条件成立：

(1) $s_1 \stackrel{a}{\Longrightarrow} s_1', a \in \text{Obs} \;\Rightarrow\; \exists s_2' \in S, s_2 \stackrel{a}{\Longrightarrow} s_2'$ 并且$s_1'\widetilde{\mathcal{B}}s_2'$；

(2) 对所有的 $\widetilde{\mathcal{B}}$ 等价类 C，且 $C \neq [s_1]_{\widetilde{\mathcal{B}}}$，有 $s_1 \stackrel{\tau}{\Longrightarrow} s_1', s_1' \stackrel{\tau}{\not\longrightarrow} \Rightarrow \exists s_2' \in S, s_2 \stackrel{\tau}{\Longrightarrow} s_2', s_2' \stackrel{\tau}{\not\longrightarrow}$ 并且$\boldsymbol{R}(s_1', C) = \boldsymbol{R}(s_2', C)$.

如果存在一个 S 上的弱互模拟等价关系 $\widetilde{\mathcal{B}}$，使得 $s_1\widetilde{\mathcal{B}}s_2$，则 s_1 与 s_2 称为弱互模拟的，记为 $s_1 \approx s_2$.

在弱互模拟等价关系的定义中，对动作转移满足的条件修改为与经典的 LTS 上的弱互模拟等价关系一样的只考虑可见动作 Obs，把内部动作都抽象掉了，因而条件 (1) 与经典的功能模型上的弱互模拟等价关系是一致的. 对于马尔可夫转移，除了如强互模拟中考虑的最大前进假设之外，另外还放松了累加转移率的计算，即在条件 (2) 中，不要求对所有的等价类 C 都要计算累加转移率，与强互模拟等价要求相比，忽略了等价状态到自身等价类的累加转移率的计算. 这一考虑是与传统的弱互模拟等价关系类似的，直观上，可以把在同一等价类中的马尔可夫转移看成是内部行为，因此到自身的转移率之和是不需要计算的.

例 4.2.4 与图4.1相对应，可以用图4.3给出一个弱互模拟等价的例子，同样，相同阴影填充的状态是弱互模拟的状态.

可以看到，图中用（▨）填充的状态是弱相互模拟的，它们之间的行为都是被看作内部的，或者说是外部不关心的. 因此它们内部之间可以任意执行内部动作 τ，并且它们到内部的累加转移率也不必考虑是否相等.

与强互模拟等价关系一样，在弱互模拟等价关系下，也可以得到一个压缩的 IMC，如图 4.4 所示. 这里，填充为（▨）的状态由原图相应填充的状态压缩而成，可见，从这个压缩的 IMC 来看，状态内部的行为都被抽象掉了，在状态（▨）内，系统是怎么运行的在外面看来是不可见的. 因此，弱互模拟等价关系很好地把系统的内部行为屏蔽掉了.

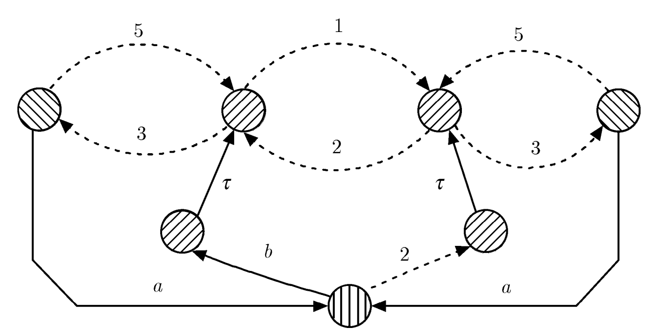


图 4.3 弱互模拟等价关系图例

观察图 4.1 中的强互模拟等价与图 4.3 中的弱互模拟等价，可以得出一个明显

的结论，即图 4.1 中的强互模拟等价的状态同样是弱相互模拟的（根据定义 4.2.3），而图 4.3 中的弱互模拟等价的状态却不满足强互模拟等价的定义. 由此可以得到下面的命题.

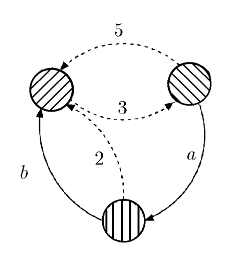


图 4.4 弱互模拟等价关系状态空间压缩图示

命题 4.2.5 对任意IMC中的两个状态 $s_1, s_2 \in S$，有

$$s_1 \sim s_2 \Rightarrow s_1 \approx s_2.$$

证明 由定义直接可得.

4.3 模拟前序关系

在分支时间的等价关系中，除了互模拟等价关系之外，还有一种重要的关系叫模拟关系. 与互模拟主要的不同是，模拟关系是一种单方向的模拟，即若系统 P_1 与 P_2 具有模拟关系，则 P_1 可以模拟 P_2 的行为，而反过来是不能保证的，P_1 中的某些行为可以是不能被 P_2 模拟的. 因此，单纯的模拟关系实际上并不是等价关系（没有对称性），而是一种前序关系（preorder）（只满足自反性和传递性）. 在传统的功能模型中，这种关系是通过只定义单方向的动作匹配实现的. 然而，在随机系统下，转移关系与一个随机分布相关，因此模拟关系必须从状态上的定义转换到随机分布上来. 这个问题采用与文献 [65] 同样的权函数的概念来帮助解决.

令 Dist(S) 表示集合 S 上的所有分布函数的集合，即

$$\text{Dist}(S) = \left\{ \mu(s) | \mu : S \to [0,1], \quad \sum_{s \in S} \mu(s) \leqslant 1 \right\},$$

其中，若 $\sum\limits_{s \in S} \mu(s) = 1$，则称分布函数 $\mu(s)$ 是随机的（stochastic），否则称为子随机的（sub-stochastic）.

定义 4.3.1 设 $\mu, \mu' \in \text{Dist}(S)$，$R \subseteq S \times S$ 是一个关系. μ 与 μ' 之间关于 R 的一个**权函数**定义为 $\Delta : S \times S \to [0,1]$，并且满足：

(1) $\Delta(s, s') > 0 \Rightarrow sRs'$;

(2) $\forall s \in S,\ \mu(s) = K_1 \cdot \sum_{s' \in S} \Delta(s, s')$;

(3) $\forall s' \in S,\ \mu'(s') = K_2 \cdot \sum s \in S\Delta(s, s')$,

其中 $K_1 = \sum_{s \in S} \mu(s), K_2 = \sum_{s' \in S} \mu'(s')$. 记 $\mu \sqsubseteq_R \mu'$，如果在 μ 与 μ' 之间存在关于 R 的一个权函数.

上述定义中, 关系 $\sqsubseteq_R$ 在 μ, μ' 均为随机分布 (即 $K_1 = K_2 = 1$) 且 R 是对称的情况下是对称的 [17]. 一个权函数实际上是定义在 $S \times S$ 上的一个概率分布，满足将 (s, s') 用 R 关联 (sRs') 的概率为 1，另外，在 R 中选择一个元素，其第一个分量为 s 的概率是 $\mu(s)$，同样，在 R 中选择出第二个分量为 s' 的概率是 $\mu'(s')$.

权函数的作用是将分布函数 μ 和 μ' 相互联系起来, 直观上, 它显示出概率 $\mu(s)$ 是怎么被分配到与 s 相联系的 s' 上，并且使得 $\mu'(s')$ 等于所有它从 Δ 得到的分配的和的. 为了将其应用到 IMC 随机行为的模拟定义上，需要对 IMC 的马尔可夫转移找到一个分布函数刻画，即找到一个从给定状态出发的转移概率分布. 为此，定义 IMC 上的状态转移分布函数如下：

定义 4.3.2 设 $s \in S$, 从 s 出发经过一个马尔可夫转移到达下一状态的概率分布定义为

$$\boldsymbol{P}(s, s') = \begin{cases} \dfrac{\boldsymbol{R}(s, s')}{E(s)}, & \text{当 } E(s) \neq 0 \text{时}, \\ 0, & \text{当 } E(s) = 0 \text{时}. \end{cases} \quad (s' \in S).$$

简记为 $\boldsymbol{P}(s, \cdot)$.

实际上，若把 $\boldsymbol{P}(s, s')$ 看成一个矩阵，它就是与 IMC 的 CTMC 部分相关的内嵌的 DTMC 的转移概率矩阵（见第二章相关定义）.

4.3.1 强模拟前序关系

有了以上概念，就可以定义强模拟前序关系如下：

定义 4.3.3 一个定义在 S 上的前序关系 $\mathcal{S}$ 称为一个**强模拟关系**，当且仅当对所有的 $s_1\mathcal{S}s_2$，以下条件成立：

(1) $s_1 \xrightarrow{a} s_1', a \in \text{Act} \ \Rightarrow\ \exists s_2' \in S, s_2 \xrightarrow{a} s_2'$，并且 $s_1'\mathcal{S}s_2'$;

(2) $s_1 \not\xrightarrow{\tau} \ \Rightarrow\ \boldsymbol{P}(s_1, \cdot) \sqsubseteq_{\mathcal{S}} \boldsymbol{P}(s_2, \cdot)$ 并且 $E(s_1) \leqslant E(s_2)$.

如果存在 S 上的一个强模拟关系 $\mathcal{S}$, 使得 $s_1\mathcal{S}s_2$，则称 s_2 强模拟 s_1，记为 $s_1 \precsim s_2$.

定义中的条件 (1) 仍然是对动作转移的要求，由于 $\mathcal{S}$ 不再是等价关系，因此由条件 (1) 不能得出对 s_2 所有能执行的动作 s_1 均能有相应的动作匹配，即 s_1 不一定可以强模拟 s_2. 在条件 (2) 中除了要求 $\boldsymbol{P}(s_1, \cdot)$ 与 $\boldsymbol{P}(s_2, \cdot)$ 满足关系 $\sqsubseteq_{\mathcal{S}}$ 之外，还要求 $E(s_1) \leqslant E(s_2)$，由于 $E(s)$ 反映了系统在 s 上的平均停留时间，因此该条件的直观含义就是从 s_2 出发的马尔可夫转移至少要与从 s_1 出发的一样快，即不能慢于 s_1.

例 4.3.4 图4.5给出了一个IMC上的强模拟关系的例子. 如图所示, 有 $s_1 \precsim s_2$. 其中, 动作转移的模拟是显然的, 因为 $s_1 \xrightarrow{a} u_3, s_2 \xrightarrow{a} v_3$ 且 $u_3 \precsim v_3$. 对于马尔可夫转移, 则有 $E(s_1) = 2 < E(s_2) = 3, \boldsymbol{P}(s_1, u_1) = \boldsymbol{P}(s_1, u_2) = \dfrac{1}{2} = \dfrac{3}{6}, \boldsymbol{P}(s_2, v_1) = \dfrac{1}{3} = \dfrac{2}{6}, \boldsymbol{P}(s_2, v_2) = \dfrac{2}{3} = \dfrac{4}{6}$, 注意 $\boldsymbol{P}(s_1, \cdot)$ 与 $\boldsymbol{P}(s_2, \cdot)$ 均是随机分布, 因此 $K_1 = K_2 = 1$. 而 $\boldsymbol{P}(s_1, \cdot)$ 与 $\boldsymbol{P}(s_2, \cdot)$ 之间的权函数则定义为 $\Delta(u_1, v_1) = \dfrac{2}{6}, \Delta(u_1, v_2) = \dfrac{1}{6}, \Delta(u_2, v_2) = \dfrac{3}{6}$. 为了验证 $\boldsymbol{P}(s_1, \cdot) \sqsubseteq_{\mathcal{S}} \boldsymbol{P}(s_2, \cdot)$, 根据权函数的定义, 计算

$$\boldsymbol{P}(s_1, u_1) = \sum_{v_i} \Delta(u_1, v_i) = \frac{1}{2}, \quad \boldsymbol{P}(s_1, u_2) = \sum_{v_i} \Delta(u_2, v_i) = \frac{1}{2},$$

$$\boldsymbol{P}(s_2, v_1) = \sum_{u_i} \Delta(u_i, v_1) = \frac{1}{3}, \quad \boldsymbol{P}(s_2, v_2) = \sum_{u_i} \Delta(u_i, v_2) = \frac{2}{3},$$

符合强模拟关系中随机分布应满足的条件.

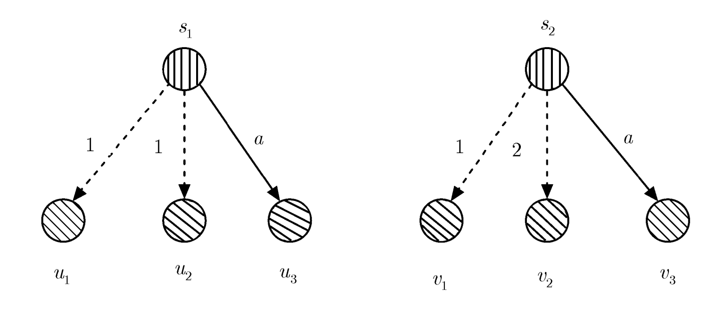


图 4.5 强模拟关系图例

关于强模拟关系与强互模拟等价关系之间的联系, 有以下的命题

命题 4.3.5 对于任意IMC及 $s_1, s_2 \in S$, 有

(1) $s_1 \sim s_2 \Rightarrow s_1 \precsim s_2$;

(2) $\precsim \cap \precsim^{-1} \Leftrightarrow \sim$.

证明 (1) 令 $\mathcal{B}$ 是 S 上的一个强互模拟关系, 需要证明 $\mathcal{B}$ 也是 S 上的强模拟关系.

对动作转移, 根据定义 4.2.1 中的条件 (1), $s_1 \sim s_2$ 意味着

$$\forall a \in \text{Act}, s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2' \in S, s_2 \xrightarrow{a} s_2' \text{ 且} s_1' \sim s_2'.$$

因此同样满足定义 4.3.3 中的条件 (1).

对于马尔可夫转移, 根据定义 4.2.1 中的条件 (2), 有 $\boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C), C \in S/\mathcal{B}$, 因此,

$$E(s_1) = \sum_{C \in S/\mathcal{B}} \boldsymbol{R}(s_1, C) = \sum_{C \in S/\mathcal{B}} \boldsymbol{R}(s_2, C) = E(s_2).$$

接下来只需证明 $\boldsymbol{P}(s_1,\cdot) \sqsubseteq_{\mathcal{B}} \boldsymbol{P}(s_2,\cdot)$ 即可. 定义权函数如下：

$$\Delta(s_1',s_2') = \begin{cases} \dfrac{\boldsymbol{P}(s_1,s_1')\cdot \boldsymbol{P}(s_2,s_2')}{\boldsymbol{P}(s_1,[s_1']_{\mathcal{B}})}, & \text{若} s_1' \sim s_2', \\ 0, & \text{否则}, \end{cases}$$

其中, $\boldsymbol{P}(s_1,[s_1']_{\mathcal{B}}) = \sum_{s_1' \in [s_1']_{\mathcal{B}}} \boldsymbol{P}(s_1,s_1')$. 则对所有 $s_1' \in S$, 有

$$\begin{aligned} \sum_{s_2' \in S} \Delta(s_1',s_2') &= \sum_{s_1' \mathcal{B} s_2'} \frac{\boldsymbol{P}(s_1,s_1')\cdot \boldsymbol{P}(s_2,s_2')}{\boldsymbol{P}(s_1,[s_1']_{\mathcal{B}})} \\ &= \frac{\boldsymbol{P}(s_1,s_1')}{\boldsymbol{P}(s_1,[s_1']_{\mathcal{B}})} \cdot \sum_{s_1' \mathcal{B} s_2'} \boldsymbol{P}(s_2,s_2') \\ &= \frac{\boldsymbol{P}(s_1,s_1')}{\boldsymbol{P}(s_1,[s_1']_{\mathcal{B}})} \cdot \boldsymbol{P}(s_2,[s_1']_{\mathcal{B}}) \\ &= \boldsymbol{P}(s_1,s_1'). \end{aligned}$$

同理, 对所有 $s_2' \in S$, 有

$$\sum_{s_1' \in S} \Delta(s_1',s_2') = \boldsymbol{P}(s_2,s_2').$$

根据定义, $\boldsymbol{P}(s_1,\cdot) \sqsubseteq_{\mathcal{B}} \boldsymbol{P}(s_2,\cdot)$ 成立. 因此, $\mathcal{B}$ 也是 S 上的强模拟关系.

(2) 由于 $\precsim$ 是自反和传递的, 所以 $\precsim \cap \precsim^{-1}$ 是自反、传递和对称的, 即它是一个等价关系.

对于动作转移, $\precsim \cap \precsim^{-1}$ 与 $\sim$ 的等价性是显然的, 只需要证明 $\precsim \cap \precsim^{-1}$ 与 $\sim$ 的马尔可夫转移满足的条件也是等价的即可.

由 (1) 可得 $\sim \Rightarrow \precsim$ 且 $\sim \Rightarrow \precsim^{-1}$, 因为 $\sim$ 是对称的. 因此有 $\sim \Rightarrow \precsim \cap \precsim^{-1}$.

要证 $\precsim \cap \precsim^{-1} \Rightarrow \sim$, 就需要证明 $\precsim \cap \precsim^{-1}$ 也是一个强互模拟等价关系. 设 $s_1 \precsim \cap \precsim^{-1} s_2$, 则 $s_1 \precsim s_2$ 并且 $s_2 \precsim s_1$, 我们证明 s_1 与 s_2 是强互模拟的. 显然, $\precsim \cap \precsim^{-1}$ 也是一个强模拟关系, 因此存在一个权函数 Δ, 满足：

(1) $\boldsymbol{P}(s_1,s_1') = \sum_{s\in S} \Delta(s_1',s)\ \forall s_1' \in S$;

(2) $\boldsymbol{P}(s_2,s_2') = \sum_{s\in S} \Delta(s,s_2')\ \forall s_2' \in S$;

(3) $\Delta(s_1',s_2') > 0 \Rightarrow s_1 \precsim \cap \precsim^{-1} s_2$.

则对任意由 $\precsim \cap \precsim^{-1}$ 导出的等价类 $C \in S/\precsim \cap \precsim^{-1}$, 有

$$\boldsymbol{P}(s_1,C) = \sum_{s_1' \in C} \boldsymbol{P}(s_1,s_1') = \sum_{s_1' \in C}\sum_{s\in S} \Delta(s_1',s) = \sum_{s_1' \in C}\sum_{s\in C} \Delta(s_1',s),$$

其中最后一个等式是因为 $\Delta(s_1',s) > 0$ 当且仅当 s 与 s_1' 在同一等价类中. 同理

$$\boldsymbol{P}(s_2,C) = \sum_{s_2' \in C} \boldsymbol{P}(s_2,s_2') = \sum_{s_2' \in C}\sum_{s\in S} \Delta(s,s_2') = \sum_{s_2' \in C}\sum_{s\in C} \Delta(s,s_2').$$

因此，对任意 $C \in S/\precsim \cap \precsim^{-1}$，都有 $\boldsymbol{P}(s_1, C) = \boldsymbol{P}(s_2, C)$. 又由 $\precsim \cap \precsim^{-1}$ 可知 $E(s_1) = E(s_2)$，且 $\boldsymbol{P}(s, C) = \boldsymbol{R}(s, C)/E(s)$，所以有

$$\boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C).$$

这表明 s_1 与 s_2 是强互模拟的.

由命题 4.3.5 可知，$\precsim \cap \precsim^{-1}$ 是一个等价关系，称之为强模拟等价关系，它实际上与强互模拟等价关系是一致的.

4.3.2 弱模拟前序关系

上一节定义的强模拟关系在很大程度上与强互模拟等价关系是一致的，因此并不是非常有用. 这一节将遵循在互模拟等价关系定义中同样的思想，给出一个弱模拟前序关系的定义，它是对强模拟前序关系做了更精心地修改得来的.

定义 4.3.6 一个定义在 S 上的前序关系 $\widetilde{\mathcal{S}}$ 称为一个**弱模拟关系**，当且仅当对所有的 $s_1\widetilde{\mathcal{S}}s_2$，以下条件成立：

(1) $s_1 \stackrel{a}{\Longrightarrow} s_1', a \in \mathrm{Obs} \;\Rightarrow\; \exists s_2' \in S, s_2 \stackrel{a}{\Longrightarrow} s_2'$ 且$s_1'\widetilde{\mathcal{S}}s_2'$;

(2) $s_1 \stackrel{\tau}{\Longrightarrow} s_1', s_1' \stackrel{\tau}{\not\longrightarrow} \;\Rightarrow\; \exists s_2' \in S, s_2 \stackrel{\tau}{\Longrightarrow} s_2', s_2' \stackrel{\tau}{\not\longrightarrow}$ 且存在权函数 $\Delta : S \times S \to [0,1], \delta_i : S \to [0,1]$ 与集合 $U_i, V_i \subseteq S (i = 1, 2)$：

$$U_i = \{u_i \in S \mid \boldsymbol{R}(s_i', u_i) > 0 \wedge \delta_i(u_i) > 0\} \text{ 和}$$
$$V_i = \{v_i \in S \mid \boldsymbol{R}(s_i', v_i) > 0 \wedge \delta_i(v_i) < 1\}$$

满足:

- $\forall v_1 \in V_1,\ v_1\widetilde{\mathcal{S}}s_2'$ 且$\forall v_2 \in V_2,\ s_1'\widetilde{\mathcal{S}}v_2$,
- $\Delta(u_1, u_2) > 0 \;\Rightarrow\; u_1 \in U_1, u_2 \in U_2$ 且 $u_1\widetilde{\mathcal{S}}u_2$,
- $\forall w \in S$,
  $K_1 \cdot \sum_{u_2 \in U_2} \Delta(w, u_2) = \delta_1(w) \cdot \boldsymbol{P}(s_1', w)$,
  $K_2 \cdot \sum_{u_1 \in U_1} \Delta(u_1, w) = \delta_2(w) \cdot \boldsymbol{P}(s_2', w)$,
- $K_1 \cdot E(s_1') \leqslant K_2 \cdot E(s_2')$,

其中 $K_i = \sum_{u_i \in U_i} \delta_i(u_i) \cdot \boldsymbol{P}(s_i', u_i)$, $i = 1, 2$. 如果存在 S 上的一个弱模拟关系 $\widetilde{\mathcal{S}}$，使得 $s_1\widetilde{\mathcal{S}}s_2$，则称 s_2 弱模拟 s_1，记为 $s_1 \precapprox s_2$.

定义中的条件 (1) 仍然与传统的功能模型上的弱模拟关系定义保持一致，即只要求单方向上外部可见动作的模拟，而不需要考虑内部动作的执行. 条件 (2) 是针对系统的随机行为做的要求，它的含义可以用图 4.6 来说明.

图中显示了两个弱模拟的状态 ($s_1 \precapprox s_2$) 应满足的条件. 直观上，s_1 与 s_2 的马尔可夫后继状态都根据函数 δ_i 划分成了两个子集，分别用 U_i 和 V_i ($i = 1, 2$) 表示 (注意集合 U_i 与 V_i 之间并不一定是没有交集的，这里为了说明方便，我们画的是不相交的集合）. 其中集合 V_i 的划分要满足条件：任何 V_2 中的状态都弱模拟 s_1,

并且 s_2 弱模拟任何 V_1 中的状态. 因此, 从 s_i 到 V_i 的马尔可夫转移看成是"内部"的, 即不可见的, 并且从 s_i 到 V_i 的转移总概率是 $1-K_i$. 而集合 U_i 的划分要求存在一个权函数 Δ 来关联两个集合中的状态, 实际上, Δ 关联的是两个从 s_i 到 U_i 的条件转移分布 $\delta_i(\cdot)\cdot \boldsymbol{P}(s_i,\cdot)/K_i$, 因此, 从 s_i 到 U_i 的马尔可夫转移看成是外部可见的, 并且从 s_i 到 U_i 的转移总概率是 K_i. 最后, 条件 $K_1\cdot E(s_1)\leqslant K_2\cdot E(s_2)$ 与强模拟前序关系类似, 要求从 s_2 到 U_2 的马尔可夫转移必须至少与从 s_1 到 U_1 的转移一样快.

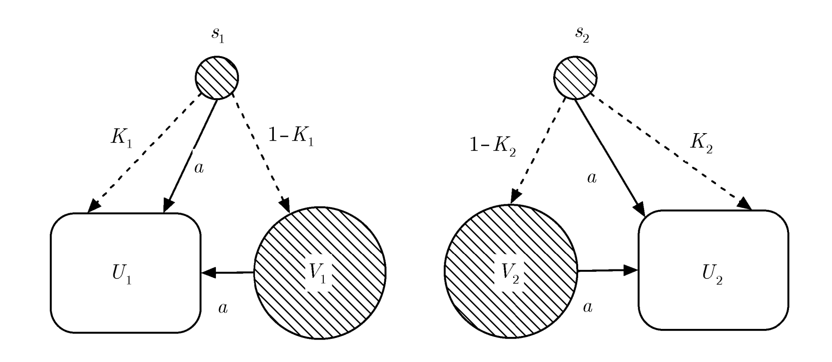


图 4.6 弱模拟前序关系说明图示

例 4.3.7 考虑图4.7所示的IMC, 有 $s_1 \precsim\!\!\approx s_2$. 对于动作转移, 弱模拟的条件是很容易验证的, 对于马尔可夫转移, s_1 的后继集合划分为

$$U_1=\{s_2\},\quad V_1=\{s_3\}.$$

即由 $\delta_1(s_2)=1, \delta_1(s_3)=0$ 得来. s_2 的后继集合划分为

$$U_2=\{s_1\},\quad V_2=\{s_3\}.$$

即由 $\delta_2(s_1)=1, \delta_2(s_3)=0$ 得来. 由此可得 $K_1=\dfrac{2}{3}, K_2=\dfrac{1}{3}$. $U_1\times U_2$ 上的权函数为 $\Delta(s_2,s_1)=1$, 满足

$$K_1\cdot\sum_{u_2\in U_2}\Delta(s_2,u_2)=\frac{2}{3}=\delta_1(s_2)\cdot\boldsymbol{P}(s_1,s_2),$$

$$K_2\cdot\sum_{u_1\in U_1}\Delta(u_1,s_1)=\frac{1}{3}=\delta_2(s_1)\cdot\boldsymbol{P}(s_2,s_1).$$

因此, 定义4.3.6中的条件(2)也是满足的.

与互模拟等价关系类似, 对于强模拟和弱模拟关系, 有以下命题

命题 4.3.8 对任意IMC 及 $s_1, s_2 \in S$，有

(1) $s_1 \precsim s_2 \ \Rightarrow\ s_1 \precapprox s_2$;

(2) $\precapprox \cap \precapprox^{-1} \Leftrightarrow \approx$.

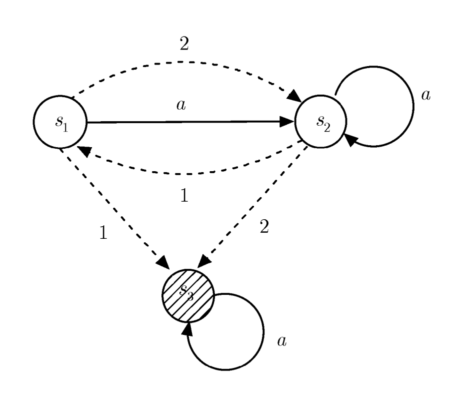


图 4.7 弱模拟前序关系图例

证明 (1) 由强模拟前序关系的定义，强模拟前序关系的动作转移满足的条件显然也满足弱模拟前序关系的定义，对于马尔可夫转移，在强模拟前序关系定义中，可以把 s_1 与 s_2 的所有马尔可夫后继看成都是集合 U_i 的元素，即都是外部可见的转移，这样，强模拟前序关系就可以看成是弱模拟前序关系中 $V_i = \varnothing$ 的一个特例. 容易验证，强模拟前序关系满足弱模拟前序在这一特例下的定义，因此，$s_1 \precsim s_2 \ \Rightarrow\ s_1 \precapprox s_2$.

(2) 对于动作转移，$\precapprox \cap \precapprox^{-1}$ 与 $\approx$ 的一致性是显然的，只需要证明 $\precapprox \cap \precapprox^{-1}$ 与 $\approx$ 在马尔可夫转移条件下是一致的即可.

(i) 必要性. 设 $s_1 \approx s_2$，则对任意 $C \in S/\approx$ 且 $C \neq [s_1]_\approx$，有 $\boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C)$. 在定义 4.3.6 中，令 $V_1 = V_2 = [s_1]_\approx, U_i = S - V_i$，即将 s_1 到与其处于同一等价类中的状态的马尔可夫转移看成是内部转移，到其他等价类的转移都是外部转移. 这样就有

$$\forall u_i \in U_i, \quad \delta_i(u_i) = 1, \quad \forall v_i \in V_i, \quad \delta_i(v_i) = 0.$$

容易验证 $\approx$ 满足在这种条件下 $\precapprox$ 的定义，因此，$s_1 \precapprox s_2$. 由 $\approx$ 的对称性，同样可以得到 $s_1 \precapprox^{-1} s_2$，这样就证明了 $\approx \Rightarrow \precapprox \cap \precapprox^{-1}$.

(ii) 充分性. 令 $R = \precapprox \cap \precapprox^{-1}$，需要证明 R 是一个弱互模拟等价关系，即对 $s_1 R s_2$，下式成立：

$$\boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C), \quad \forall C \in S/R, \quad C \neq [s_1]_R. \qquad (*)$$

设 $s_1 R s_2$ 且 $s_1, s_2 \notin U \subseteq S$. 定义集合 U 关于 $\precapprox$ 的上封闭集合与下封闭集合如下：

$$U = U\uparrow = \{w \mid u \precapprox w, \quad u \in U\},$$

$$U = U\downarrow = \{w \mid w \precsim u, \quad u \in U\}.$$

证明无论 $U = U\uparrow$ 或是 $U = U\downarrow$，都有 $\boldsymbol{R}(s_1, U) = \boldsymbol{R}(s_2, U)$. 这里只给出 $U = U\uparrow$ 时的证明, $U = U\downarrow$ 的情况可以同理证明.

令 $\mathrm{Post}(s_1) = \{s \in S \mid \boldsymbol{R}(s_1, s) > 0\} \subseteq s_2\downarrow = \{s \in S \mid s \precsim s_2\}$, 则 $\mathrm{Post}(s_1) \cap U = \varnothing$，否则 $s_2 \in U\uparrow = U$. 因此，有

$$\boldsymbol{R}(s_1, U) = 0 \leqslant \boldsymbol{R}(s_2, U).$$

现在，假设存在定义 4.3.6 中的 $\delta_i, U_i, V_i, K_i, \Delta$，其中，$K_1 > 0$. 由于 $K_1 \cdot E(s_1) \leqslant K_2 \cdot E(s_2)$，所以，$K_2 > 0$. 更进一步，有

$$v \in V_1 \Rightarrow vRs_2 \Rightarrow v \precsim s_2.$$

由此可知 $v \notin U$, 因为否则 $v \in V_1 \cap U$, 将导致 $s_2 \in U = U\uparrow$. 因此 $\mathrm{Post}(s_1) \cap U \subseteq U_1$ 且对所有 $u \in \mathrm{Post}(s_1) \cap U$，$\delta_1(u) = 1$. 现在，有如下计算

$$\begin{aligned}
\boldsymbol{R}(s_1, U) &= E(s_1) \cdot \sum\nolimits_{u \in U} \boldsymbol{P}(s_1, u) \\
&= E(s_1) \cdot \sum\nolimits_{u \in U} \sum\nolimits_{u_2 \in S} \Delta(u, u_2) \cdot K_1 \\
&= E(s_1) \cdot K_1 \cdot \sum\nolimits_{u \in U} \sum\nolimits_{u_2 \in U} \Delta(u, u_2) \\
&= E(s_1) \cdot K_1 \cdot \sum\nolimits_{u_2 \in U} \sum\nolimits_{u \in U} \Delta(u, u_2) \\
&\leqslant E(s_1) \cdot K_1 \cdot \sum\nolimits_{u_2 \in U} \sum\nolimits_{u \in S} \Delta(u, u_2) \\
&= E(s_1) \cdot K_1 \cdot \sum\nolimits_{u_2 \in U} \delta_2(u_2) \cdot \frac{\boldsymbol{P}(s_2, u_2)}{K_2} \\
&\leqslant \frac{E(s_1) \cdot K_1}{K_2} \cdot \sum\nolimits_{u_2 \in U} \boldsymbol{P}(s_2, u_2) \\
&\leqslant E(s_2) \cdot \boldsymbol{P}(s_2, U) \\
&= \boldsymbol{R}(s_2, U).
\end{aligned}$$

类似地，可以证明 $\boldsymbol{R}(s_2, U) \leqslant \boldsymbol{R}(s_1, U)$，由此可得 $\boldsymbol{R}(s_2, U) = \boldsymbol{R}(s_1, U)$.

利用上面的结果，现在可以证明 $(*)$ 式了. 令 $C, B \in S/R, B \neq C$ 且 $s_1, s_2 \in B$.

情形 1 $C \not\precsim B$，即不存在 $s \in C$ 和 $s' \in B$ 且 $s \precsim s'$. 则 $C\uparrow$ 与 $C\uparrow \setminus C$ 都不包含 s_1 和 s_2. 有

$$\boldsymbol{R}(s_1, C\uparrow) = \boldsymbol{R}(s_2, C\uparrow),$$

$$\boldsymbol{R}(s_1, C\uparrow) - \boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_1, C\uparrow \setminus C) = \boldsymbol{R}(s_2, C\uparrow \setminus C) = \boldsymbol{R}(s_2, C\uparrow) - \mathbf{R}(s_2, C).$$

因此，$\boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C)$.

情形 2 $C \precsim B$，即对所有 $s \in C$ 和 $s' \in B$，有 $s \precsim s'$. 则 $C\downarrow \cap B = \varnothing$，因为 $C \cap B = \varnothing$. 剩下 $\boldsymbol{R}(s_1, C) = \boldsymbol{R}(s_2, C)$ 的证明就如情形 1 中的证明，只需将 $C\uparrow$ 替换成 $C\downarrow$ 即可.

4.4 逻辑特征

本节将研究前面定义的互模拟等价关系及模拟前序关系的逻辑特征. 所谓一个 (等价) 关系的逻辑特征, 指的两个相互关联的状态所满足的逻辑公式的特征. 我们将用 IMC 的刻画逻辑 aCSL 来反映 IMC 上的分支时间等价关系的特征.

4.4.1 互模拟关系的逻辑特征

关于互模拟等价关系的逻辑特征, 在经典的功能模型上定义的强互模拟等价已经被证明与 CTL 等价是一致的 [26], 即相互强模拟的两个状态满足同样的 CTL 公式, 反之亦然. 而对于性能模型, 已经证明了 CTMC 上的强互模拟等价与 CSL 等价是一致的 [9,39]. 称这种性质为强保持性, 即若 $s \sim s'$, 则对所有公式 Φ, 有 $s \models \Phi \;\Leftrightarrow\; s' \models \Phi$. 对于弱互模拟关系, 也有类似的性质, 只是弱互模拟不保持所有的公式, 而只保持那些没有 X (下一步) 算子的公式 ($\mathrm{CTL}_{\backslash X}$ 和 $\mathrm{CSL}_{\backslash X}$). 本节将证明我们在 IMC 上定义的互模拟等价关系同样具有类似的特征.

为简便起见, 用 $\equiv_L$ 表示关于某个逻辑 L 等价, 即

$$s_1 \equiv_L s_2 \text{ 当且仅当} \forall \Phi \in L, s_1 \models \Phi \Leftrightarrow s_2 \models \Phi.$$

令 aCSL^- 表示 aCSL 的一个子集, 其中在路径公式 $\Phi_A \mathcal{U}^{<t}{}_B \Phi$ 中限制 $\tau \notin B$, 这样, 在 aCSL^- 中, 单独执行一个内部动作 τ 的路径公式是不允许的, 因此它实际上是与 $\mathrm{CTL}_{\backslash X}$ 和 $\mathrm{CSL}_{\backslash X}$ 相对应的一个子集公式.

下面的定理揭示了在 IMC 中定义的互模拟等价关系与 aCSL 逻辑等价性之间的关系.

定理 4.4.1 对于任意一个IMC, $s_1, s_2 \in S$, 有

(1) $s_1 \sim s_2 \Leftrightarrow s_1 \equiv_{\mathrm{aCSL}} s_2$;

(2) $s_1 \approx s_2 \Leftrightarrow s_1 \equiv_{\mathrm{aCSL}^-} s_2$.

证明 (1) 充分性. 需要证明如果 $s_1 \sim s_2$, 则对任何 aCSL 状态公式 Φ, 都有 $s_1 \models \Phi$ 和 $s_2 \models \Phi$ 成立.

对 Φ 用归纳法来获得证明.

(i) $\Phi = \mathrm{true}$. 根据定义, 任何状态都满足公式 $\Phi = \mathrm{true}$, 因此结论成立.

(ii) 假设对任何 aCSL 状态公式 Φ, 结论成立, 即有 $s_1 \sim s_2 \Rightarrow s_1 \models \Phi \wedge s_2 \models \Phi$, 则对公式 $\neg\Phi$ 和 $\Phi \wedge \Phi$, 结论也显然成立. 唯一需要证明的是对公式 $\Phi = \mathcal{P}_{\bowtie p}(\varphi)$, 结论也成立, 也就是要证明 $s_1 \models \mathcal{P}_{\bowtie p}(\varphi) \wedge s_2 \models \mathcal{P}_{\bowtie p}(\varphi)$. 根据该公式的语义定义, 只需要证明对任意 aCSL 路径公式 φ, 都有

$$\Pr\{\sigma \in \mathrm{Path}(s_1) \mid \sigma \models \varphi\} = \Pr\{\sigma \in \mathrm{Path}(s_2) \mid \sigma \models \varphi\}.$$

路径公式有两种, 只证明公式 $\varphi = \Phi_1 \; {}_A\mathcal{U}^{<t}{}_B \Phi_2$, 另外一种形式的路径公式可

类似证明. 首先定义路径集合 $A_n^s(t)$ 如下:

$$\begin{aligned} A_n^s(t) = \{\sigma \in \text{Path}(s) \mid \quad & \sigma[n] \models \Phi_2 \\ & \wedge(\forall 0 \leqslant i < n-1.\sigma[i] \models \Phi_1 \wedge \sigma[i] \xrightarrow{L_A^*} \sigma[i+1]) \\ & \wedge \sigma[n-1] \models \Phi_1 \wedge \sigma[n-1] \xrightarrow{L_B} \sigma[n] \\ & \wedge \sum\nolimits_{i=0}^{n-1} \delta(\sigma, i) < t\}, \end{aligned}$$

其中, $n \geqslant 1, t > 0, s \in S$, L_A^* 与 L_B 与第三章中的定义相同. 再令

$$B_n^s(t) = \begin{cases} A_n^s(t), & n = 1, \\ A_n^s(t) \backslash \bigcup_{i=1}^n B_i^s(t), & n > 1. \end{cases}$$

直观上, $A_n^s(t)$ 是从 s 出发, 在时间 t 内经过 n 步转移到达 Φ_2 状态, 并且前 $n-1$ 步转移只执行 A 中的动作, 最后一步执行 B 中的动作转移的路径集合. 而 $B_n^s(t)$ 则是 $A_n^s(t)$ 的一个子集, 是从 s 出发, 在时间 t 内经过 n 步转移到达 Φ_2 状态, 并且没有执行 $A \cap B$ 中的动作转移的路径集合. 注意, 根据 $B_n^s(t)$ 的定义, $B_i^s(t), B_j^s(t)$ 是两两不相交的集合 $(i \neq j)$, 并且

$$\{\sigma \in \text{Path}(s) \mid \sigma \models \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2\} = \bigcup\nolimits_{i \geqslant 1} B_i^s(t).$$

因此,

$$\Pr\{\sigma \in \text{Path}(s) \mid \sigma \models \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2\} = \sum_{i=1}^{\infty} \Pr\{\sigma \in B_i^s(t)\}.$$

上面的等式就将证明目标转向了下式

$$\sum_{i=1}^{\infty} \Pr\{\sigma \in B_i^{s_1}(t)\} = \sum_{i=1}^{\infty} \Pr\{\sigma \in B_i^{s_2}(t)\}.$$

上式可以通过一个更强的结论来证明, 即

$$\forall n > 0, \Pr\{\sigma \in B_n^{s_1}(t)\} = \Pr\{\sigma \in B_n^{s_2}(t)\}. \tag{$\triangle$}$$

以下通过另外一个对 n 的归纳来证明 (△) 式.

$n = 1$. 集合 $B_1^{s_1}(t)$ 和 $B_1^{s_2}(t)$ 都包含了所有只经过一个转移到达 Φ_2 状态的路径, 且这个转移是 B 动作转移, 由 Pr 的定义, 有

$$\Pr\{\sigma \in B_1^{s_1}(t)\} = \Pr\{\sigma \in B_1^{s_2}(t)\} = 1.$$

设对任意 $n > 1, t > 0$ 且 $s_1 \sim s_2$, 都有 $\Pr\{\sigma \in B_n^{s_1}(t)\} = \Pr\{\sigma \in B_n^{s_2}(t)\}$, 则 $B_{n+1}^{s_1}(t)$ 中的路径可以通过以下两个步骤来构造: 首先从 s_1 出发, 在时间 x 内经过一个转移到达下一个仍然满足 Φ_1 的状态 u, 接下来再从 $B_n^u(t-x)$ 中选择一条路径完成构造. 这样, 根据从 s_1 出发的第一个转移, 分别讨论以下两种情况:

情形 1 第一个转移是动作转移，则根据Pr的定义

$$\Pr\{\sigma \in B_{n+1}^{s_1}(t)\} = \Pr\{\sigma \in B_n^u(t-x)\}.$$

由 $s_1 \sim s_2$ 和对 Φ 的归纳假设（强互模拟的状态满足同样的aCSL状态公式），可以从 s_2 出发找到一个相同的动作转移在时间 x 内到达状态 v，且 $v \sim u$，所以同样有

$$\Pr\{\sigma \in B_{n+1}^{s_2}(t)\} = \Pr\{\sigma \in B_n^v(t-x)\}.$$

又由对 n 的归纳假设，有

$$\Pr\{\sigma \in B_n^u(t-x)\} = \Pr\{\sigma \in B_n^v(t-x)\}.$$

因此

$$\Pr\{\sigma \in B_{n+1}^{s_1}(t)\} = \Pr\{\sigma \in B_{n+1}^{s_2}(t)\}.$$

情形 2 第一个转移是马尔可夫转移，则有以下计算

$$\begin{aligned}
\Pr\{\sigma \in B_{n+1}^{s_1}(t)\} &= \int_0^t e^{-E(s_1)\cdot x} \cdot \sum_{u\models\Phi_1} \boldsymbol{R}(s_1,u)\cdot \Pr\{\sigma \in B_n^u(t-x)\}\ dx \\
&= \int_0^t e^{-E(s_1)\cdot x} \cdot \sum_{C\in S/\sim,\ C\models\Phi_1} \sum_{u\in C} \boldsymbol{R}(s_1,u)\cdot \\
\Pr\{\sigma \in B_n^u(t-x)\}\ dx &\overset{*}{=} \int_0^t e^{-E(s_2)\cdot x} \cdot \sum_{C\in S/\sim,\ C\models\Phi_1} \sum_{u\in C} \boldsymbol{R}(s_2,u)\cdot \\
\Pr\{\sigma \in B_n^u(t-x)\}\ dx &= \int_0^t e^{-E(s_2)\cdot x} \cdot \sum_{u\models\Phi_1} \boldsymbol{R}(s_2,u)\cdot \Pr\{\sigma \in B_n^u(t-x)\}\ dx \\
&= \Pr\{\sigma \in B_{n+1}^{s_2}(t)\},
\end{aligned}$$

其中，$C \models \Phi_1$ 表示 C 中的所有状态都满足公式 Φ，这一条件由对 Φ 的归纳假设保证. 带 $*$ 号的等式成立是因为 $s_1 \sim s_2$ 可以推出 $\sum_{u\in C}\boldsymbol{R}(s_1,u) = \sum_{u\in C}\boldsymbol{R}(s_2,u)$ 和 $E(s_1) = E(s_2)$.

这样就完成了对 n 的归纳证明，即 (△) 式成立. 这也结束了对公式 $\Phi = \mathcal{P}_{\bowtie p}(\varphi)$ 的归纳证明. 至此，完成了充分性的证明.

必要性. 需要证明，如果对任意的 aCSL 状态公式 Φ，$s_1 \models \Phi$ 且 $s_2 \models \Phi$，则 $s_1 \sim s_2$. 证明过程也是对 Φ 的结构做归纳，如充分性中的证明一样，只需要对公式 $\mathcal{P}(\varphi)$ 给出归纳证明即可. 这里，同样只给出 $\varphi = \Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$ 的证明，另外一种形式的路径公式的证明是类似的.

采用前面定义的记号，根据前面分析，若 $s_1 \models \mathcal{P}(\varphi) \wedge s_2 \models \mathcal{P}(\varphi)$，则有

$$\forall n > 0, \Pr\{\sigma \in B_n^{s_1}(t)\} = \Pr\{\sigma \in B_n^{s_2}(t)\}.$$

如果 $n=1$，则 σ 只包含一个 B 转移，显然 $s_1 \sim s_2$.

如果 $n>1$，且 σ 的第一个转移是动作转移，即 s_1 与 s_2 都可以执行一个 A 转移到达下一个仍然满足 Φ_1 的状态，由归纳假设，这些满足 Φ_1 的状态都是强互模拟的；另一方面，如果 σ 的第一个转移是马尔可夫转移，则对 $\Pr\{\sigma \in B_n^{s_1}\}$ 和 $\Pr\{\sigma \in B_n^{s_2}\}$，可以类似充分性中的计算，分别得到

$$\Pr\{\sigma \in B_n^{s_1}(t)\} = \int_0^t e^{-E(s_1)\cdot x} \cdot \sum_{C\in S/\sim,\ C\models\Phi_1} \boldsymbol{R}(s_1,C)\cdot \Pr\{\sigma \in B_{n-1}^u(t-x)\}\ dx,$$

$$\Pr\{\sigma \in B_n^{s_2}(t)\} = \int_0^t e^{-E(s_2)\cdot x} \cdot \sum_{C\in S/\sim,\ C\models\Phi_1} \boldsymbol{R}(s_2,C)\cdot \Pr\{\sigma \in B_{n-1}^u(t-x)\}\ dx.$$

由此可得对任意 $C \in S/\sim$，都有 $\boldsymbol{R}(s_1,C)=\boldsymbol{R}(s_2,C)$. 因此，根据强互模拟等价的定义，有 $s_1 \sim s_2$.

(2) 由于弱互模拟等价关系中不考虑内部动作，且计算累加转移率时不需要考虑到自身等价类的转移率，而 aCSL$^-$ 公式中也不允许只有一步内部转移的路径公式，因此，可以采用类似 (1) 的证明，其中在处理动作转移时只需要忽略掉内部动作 τ 即可得到该结论.

定理 4.4.1 说明，在 IMC 上定义的互模拟等价关系具有良好的逻辑特征，两个强互模拟的状态满足相同的 aCSL 逻辑公式，因此也可以说它们是 aCSL 等价的. 而弱互模拟等价关系也可以用一个 aCSL 的子逻辑 aCSL$^-$ 来刻画，由于在弱等价关系下，系统的单步执行是不保证完全匹配的 (由于内部动作的存在)，因此，弱等价关系的刻画逻辑必须将单步执行的刻画排除掉，这就是必须采用 aCSL$^-$ 的原因.

推论 4.4.2 对任意IMC及 $s_1, s_2 \in S$，有

$$s_1 \equiv_{\text{aCSL}} s_2 \ \Rightarrow\ s_1 \equiv^-_{\text{aCSL}} s_2.$$

证明 由 $\sim \Rightarrow \approx$ 及定理 4.4.1 直接可得.

4.4.2 模拟关系的逻辑特征

上一节看到，互模拟等价关系具有很好的逻辑特征，它具有所谓的强保持性，即互模拟的状态将保证满足相同的逻辑公式. 对于模拟关系，它的这种逻辑公式的保持性不如互模拟关系强，它只对一类公式有部分的保持，例如，对于传统的 LTS 上的模拟关系 (记为 $\sqsubseteq$)，若 $s \sqsubseteq s'$，则对所有 ACTL* 公式 Φ，有 $s' \models \Phi \ \Rightarrow\ s \models \Phi$. 这里，ACTL* 公式是 CTL* 公式的一个子集，是将 CTL* 路径公式的路径量词限制为 $\forall$ 得到的[1]. 注意，在这里 $s' \not\models \Phi$ 不一定说明 $s \not\models \Phi$. 因此，称模拟关系的这种性质为弱保持性. 对于随机系统如 CTMC，也证明了 CTMC 上的模拟关系对 CSL 的一个子集 (安全性与活性公式集) 具有同样的弱保持性 [17].

[1]通常这一子集公式也称为安全性公式 (safety formula) .

本节将证明，在 IMC 上定义的模拟关系同样具有类似的弱保持性，其弱保持的是 aCSL 的一个安全性（或者活性）公式子集.

首先给出 aCSL 的安全性和活性公式的定义. 安全性与活性是两类重要的性质，广泛用于描述并发系统的关键行为. 安全性断言“坏的事情永远不会发生”，例如，在一个多任务操作系统中，系统不会发生死锁就是对系统的一个安全性做的断言. 而活性则用于表达“好的事情终究会发生”的含义，例如，在网络传输当中，每个发出的数据包最终都会被收到就是系统的一个活性特征的例子. 安全性与活性在不同的描述语言中有不同的定义，这里给出在 aCSL 刻画下的安全性与活性的定义.

定义 4.4.3 设 $p \in [0,1]$，aCSL**安全性公式**由以下语法产生：

$$\Phi ::= \mathrm{true} \mid \mathrm{false} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathcal{P}_{\leqslant p}(\neg\Phi_A \mathcal{U}^{<t}{}_B \neg\Phi) \mid \mathcal{P}_{\leqslant p}(\neg\Phi_A \mathcal{U}^{<t} \neg\Phi).$$

注意到在 aCSL 安全性公式中，不存在公式的否定形式，即 aCSL 公式都以正范式（positive normal form）来表示，并且在概率算子 $\mathcal{P}$ 中，概率界也被限制为只能是“$\leqslant p$”. 通常在表示一个安全性公式时，p 的取值都尽可能地接近 0.

例 4.4.4 公式 $\mathcal{P}_{\leqslant 0.01}(\mathrm{true}_{\{\mathrm{Run}\}} \mathcal{U}^{<100} \neg\mathrm{true})$ 是一个安全性公式的例子，它表达了系统在100个时间单位内出故障的概率至多为0.01（即“坏事永远不会发生”的概率描述）.

aCSL 活性公式的定义与安全性公式的定义类似.

定义 4.4.5 设 $p \in [0,1]$，aCSL**活性公式**由以下语法产生：

$$\Phi ::= \mathrm{true} \mid \mathrm{false} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathcal{P}_{\geqslant p}(\Phi_A \mathcal{U}^{<t}{}_B \Phi) \mid \mathcal{P}_{\geqslant p}(\Phi_A \mathcal{U}^{<t} \Phi).$$

aCSL 活性公式同样是正范式形式，且 $\mathcal{P}$ 算子中的概率界被限制为“$\geqslant p$”. 与安全性公式相反，在表示一个活性公式时，p 的取值是尽可能地接近 1.

例 4.4.6 公式 $\mathcal{P}_{\geqslant 0.99}(\mathrm{true}_{\{\mathrm{request}\}} \mathcal{U}^{<100}{}_{\{\mathrm{acknowledge}\}} \neg\mathrm{true})$ 是一个活性公式的例子，它表示如果系统中有请求，则在100个时间单位内得到应答的概率不小于0.99（即“好的事情终究会发生”的概率描述）.

实际上，aCSL 安全性公式与 aCSL 活性公式是对偶的，即对任何 aCSL 安全性公式 Φ，都存在一个活性公式与 $\neg\Phi$ 等价，反过来也一样. 例如，设 Φ 是例 4.4.4 中的安全性公式，则 $\neg\Phi = \mathcal{P}_{\geqslant 0.99}(\neg\mathrm{true}_{\{\mathrm{Run}\}} \mathcal{U}^{<100} \mathrm{true})$ 即是与 Φ 对偶的一个活性公式[2]. 因此，接下来只需要考虑安全性公式，对于活性公式，以下得到的结果也是成立的.

为了叙述简单，用 $\preccurlyeq_L$ 来表示对某个逻辑 L 的弱保持性，即

$$s_1 \preccurlyeq_L s_2 \text{ 当且仅当} \forall \Phi \in L, s_2 \models \Phi \Rightarrow s_1 \models \Phi.$$

[2]这里得到的对偶公式只是满足活性公式的定义，但其含义在原公式的上下文中并不一定表达实际的活性概念.

令 aCSL_S 表示 aCSL 安全性公式，aCSL_S^- 表示 aCSL_S 的一个子集，满足在 aCSL_S 的路径公式 $\Phi_A\mathcal{U}^{<t}{}_B\Phi$ 中，$\tau \notin B$. 因此，aCSL_S^- 与 aCSL^- 类似，它禁止了 aCSL_S^- 中单步执行一个内部动作的公式.

下面的定理给出了 IMC 上的模拟前序关系关于 aCSL 安全公式的的弱保持性质.

定理 4.4.7 对于任意一个IMC，$s_1, s_2 \in S$，有

(1) $s_1 \precsim s_2 \Rightarrow s_1 \preccurlyeq_{\mathrm{aCSL}_S} s_2$，

(2) $s_1 \precapprox s_2 \Rightarrow s_1 \preccurlyeq_{\mathrm{aCSL}_S^-} s_2$.

证明 (1) 要证明结论，即是要证明对 $s_1 \precsim s_2$，有下式成立

$$s_2 \models \Phi \ \Rightarrow \ s_1 \models \Phi,\ \forall \Phi \in \mathrm{aCSL}_S.$$

同样对公式 Φ 的结构用归纳法进行证明. 其中，对公式 ture, false,$\Phi \wedge \Phi, \Phi \vee \Phi$ 的证明是显而易见的，唯一需要证明的是概率算子 $\mathcal{P}$. 以下仍然以路径公式 $\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$ 为例给出证明，另一形式的路径公式可类似证明.

要证明 $s_2 \models \mathcal{P}_{\leqslant p}(\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2) \ \Rightarrow \ s_1 \models \mathcal{P}_{\leqslant p}(\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2)$，根据 aCSL 公式的语义，也就是要证明

$$\mathrm{Prob}(s_2, \Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2) \leqslant \mathrm{Prob}(s_1, \Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2).$$

根据定理 4.4.1 中同样的分析，只需要证明

$$\forall n > 0, \Pr\{\sigma \in B_n^{s_2}(t)\} \leqslant \Pr\{\sigma \in B_n^{s_1}(t)\} \qquad (\triangle\triangle)$$

即可. 证明与定理 4.4.1 一样是对 n 做归纳，归纳基础 $(n=1)$ 的证明与定理 4.4.1 是一样的，这里对归纳步骤给出证明.

假设对任意 $n > 1, t > 0$ 且 $s_1 \precsim s_2$，都有 $\Pr\{\sigma \in B_n^{s_2}(t)\} \leqslant \Pr\{\sigma \in B_n^{s_1}(t)\}$，则对 $B_{n+1}^{s_1}(t)$ 的构造同样分以下两种情况来证明.

情形 1 第一个转移是个动作转移. 则根据Pr的定义

$$\Pr\{\sigma \in B_{n+1}^{s_1}(t)\} = \Pr\{\sigma \in B_n^{u}(t-x)\},$$

由 $s_1 \precsim s_2$ 对 Φ 的归纳假设，可以从 s_2 出发找到一个相同的动作转移在时间 x 内到达状态 v，且 $u \precsim v$，所以同样有

$$\Pr\{\sigma \in B_{n+1}^{s_2}(t)\} = \Pr\{\sigma \in B_n^{v}(t-x)\}.$$

又由对 n 的归纳假设，有

$$\Pr\{\sigma \in B_n^{v}(t-x)\} \leqslant \Pr\{\sigma \in B_n^{u}(t-x)\},$$

因此，

$$\Pr\{\sigma \in B_{n+1}^{s_2}(t)\} \leqslant \Pr\{\sigma \in B_{n+1}^{s_1}(t)\}.$$

情形 2 第一个转移是马尔可夫转移，则有以下计算

$$\Pr\{\sigma \in B_{n+1}^{s_1}(t)\} = \int_0^t e^{-E(s_1)\cdot x} \cdot \sum_{u \models \Phi_1} \boldsymbol{R}(s_1, u) \cdot \Pr\{\sigma \in B_n^u(t-x)\}\, dx,$$

$$\Pr\{\sigma \in B_{n+1}^{s_2}(t)\} = \int_0^t e^{-E(s_2)\cdot x} \cdot \sum_{v \models \Phi_1 \wedge u \precsim v} \boldsymbol{R}(s_2, v) \cdot \Pr\{\sigma \in B_n^v(t-x)\}\, dx.$$

由 $s_1 \precsim s_2$ 有 $E(s_1) \leqslant E(s_2)$，又根据归纳假设，对 $u \precsim v$，有 $\Pr\{\sigma \in B_n^v(t-x)\} \leqslant \Pr\{\sigma \in B_n^u(t-x)\}$，所以同样有

$$\Pr\{\sigma \in B_{n+1}^{s_2}(t)\} \leqslant \Pr\{\sigma \in B_{n+1}^{s_1}(t)\}.$$

由归纳法知 (△△) 式成立，从而结论成立.

(2) 对于弱模拟关系，其证明与 (1) 类似，不同的是在处理动作转移时忽略掉内部动作，而在处理马尔可夫转移时，其行为退化成一个 CTMC，已经证明了 CTMC 上的弱模拟关系与 CSL 安全公式之间的蕴涵关系 [16]，可以利用这个结果，结合 (1) 中的证明策略得到定理中结论（其中的详细证明过程可参考文献 [16]）.

由于 aCSL_S^- 公式是 aCSL_S 的一个严格的子集，因此有以下显然的推论.

推论 4.4.8 对任意IMC及 $s_1, s_2 \in S$，有

$$s_1 \preccurlyeq_{\mathrm{aCSL}_S} s_2 \;\;\Rightarrow\;\; s_1 \preccurlyeq_{\mathrm{aCSL}_S^-} s_2.$$

4.5 小　　结

本章在 IMC 上定义了各种分支时间的等价关系，包括互模拟和模拟关系，并且研究了这些关系之间的相互联系. 由于 IMC 结合了传统的功能模型和性能模型，因此我们的目标是把经典的 LTS 上的分支时间等价关系和纯马尔可夫链上的等价关系用一个统一的框架结合起来. 由于 IMC 模型的特性，可以很容易地将这两种模型上的相关概念结合，并且仍然保持了这些等价关系在原来各自模型上良好的性质. 很显然，本章定义的 IMC 上的各种分支时间等价关系的概念是原有各自模型上相应概念的自然推广，并且与已有的定义相兼容.

除了统一两个模型上的分支时间等价关系概念之外，本章内容的另外一个突出的创新点是在 aCSL 的基础上给出了这些关系的逻辑特征. 所谓的等价关系或前序关系的逻辑特征，指的是两个相互之间有关系的状态在逻辑上具有什么联系，即满足的逻辑公式之间有什么联系. 研究结果表明，IMC 上的互模拟等价关系对 aCSL 公式具有强保持性，而模拟前序关系也对 aCSL 的安全性公式（或活性公式）具有弱保持性. 这一结果使得 IMC 上的分支时间等价关系概念延伸到了逻辑层次，为其他相关的研究工作奠定了良好的基础.

本章对各种等价关系和前序关系之间的相互联系也作了研究，得到的结果可以用图 4.8 来直观的总结. 图中 $R \longrightarrow R'$ 表示关系 R 可以导出关系 R'.

文献 [18] 研究了 DTMC 和 CTMC 上的各种分支时间等价关系的概念和相互之间的联系，图 4.8 显示的结果可以看成是文献 [18] 中得到的结果的补充和扩展. 这样，就在 IMC 框架下建立起了一个相对完整的等价关系谱. 这些分支时间等价关系之间的联系既包含了传统功能模型上分支时间等价关系的结果，也包含了性能模型上的分支时间等价关系的结果，因此可以说我们的结果将这些已有的结果很好地统一了起来.

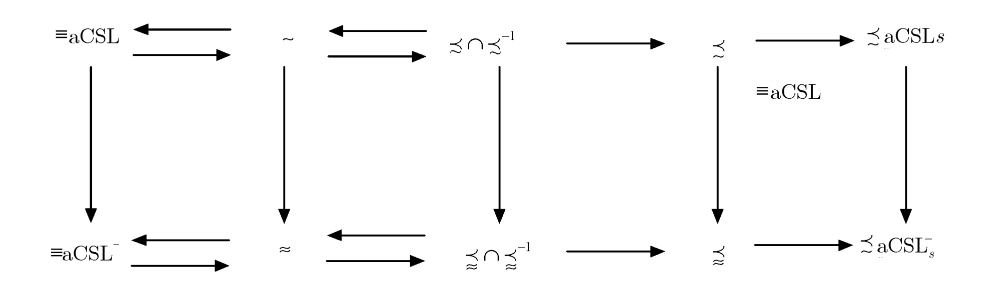


图 4.8　IMC 上的分支时间等价关系谱

根据这些等价关系的逻辑特征，将在下一章中阐明它们在解决模型检验的状态爆炸问题中的应用.

第五章　动 作 细 化

本章将从系统设计方法学的角度研究 IMC 作为一种功能与性能混合的并发系统模型的建模能力. 我们研究的是经典的“自顶向下，逐步细化”的设计思想在并发系统建模中的应用. 通过在 IMC 模型上定义动作细化（action refinement）操作，建立起 IMC 模型上的层次化设计理论框架.

5.1　概　　述

5.1.1　什么是动作细化

对于复杂并发系统的设计，通常有两种可以进行和简化设计过程的方法，一种是组合化，即将系统看作由一个个小的子模块（子系统）构成，一个大的复杂系统就可以经由这些子模块的组合来表示，这样，对复杂系统的刻画以及分析就可以模块化，每次只需要将注意力放到系统的一小部分细节上，这种模块化称之为水平模块化（horizontal modularity）. 本书前面介绍的进程代数实际上就是这种设计思想的一种体现，也是进程代数最主要的优越性体现.

图 5.1 显示了一个采用这种水平模块化（组合化）设计方法进行系统设计的例子. 在图中，整个系统由 3 个组件 C1, C1,C3 构成，采用模块化思想，可以先分别对 3 个组件进行设计，然后通过一定的协议（接口），将它们组合成一个整体. 因此，在整个设计过程中，每次只需要关注其中一个组件的功能就行了.

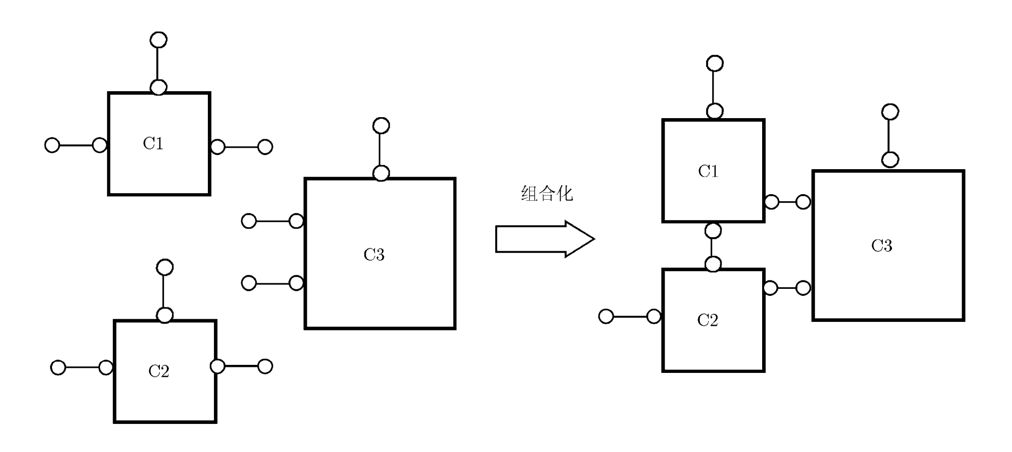


图 5.1　水平模块化 (组合化) 设计方法示意图

然而，从软件工程的观点来看，仅有组合化的设计方法在很多情况下还是不够

的，因为组合化的思想是在同一个抽象层次上的，一旦描述系统行为的动作集合确定了，则系统的抽象层次就固定了. 而在软件开发过程当中，有可能需要对概念上属于不同抽象层次的系统行为进行比较，以便验证它们本质上实现的是同一个功能. 因此，在软件工程当中通常还涉及另外一种设计方法，即将一个复杂的系统首先描述成一个简单、抽象的系统规范 (specification)，然后一步一步的细化到实际的、复杂的系统实现 (implementation)，其中对每一步的刻画与分析都是在不同的抽象层次上进行，因此可以只关注当前抽象级别上的细节. 这种设计方法与前面说到的组合化的思想完全不同，我们称之为垂直模块化 (vertical modularity)，通常也叫做层次化 (hierarchical) 设计方法. 这种设计方法在传统顺序系统的设计中得到了成功的应用，产生了著名的自顶向下的系统设计方法 (top-down system design)，如图 5.2 所示.

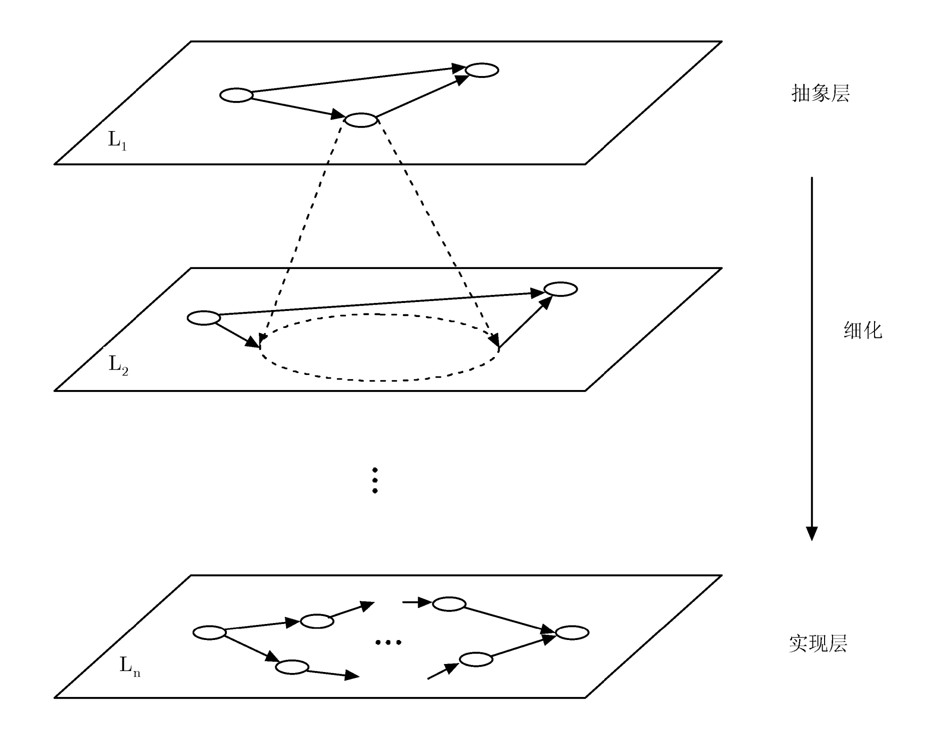


图 5.2　垂直模块化 (层次化) 设计方法示意图

在进程代数的框架下，并发系统的设计也可以采用这种自顶向下的策略，这就需要引入一种从高层次抽象转化到低层次进程描述的机制，这种机制就是动作细化. 动作细化是基于系统由其所能执行的动作来表示的一种层次化设计方法，即它主要关注系统的动作，以及如何将表示系统行为的一个抽象动作转换成一组更具体的动

作. 因此从概念上来说，只要是基于动作的系统，都可以引入动作细化的机制，使其获得层次化设计的能力.

5.1.2 动作细化的不同观点

传统的以进程代数描述的并发系统中的动作细化理论得到了多方面的研究[2~4,50,70,82,101,102]，这些研究使得人们对动作细化的概念从各种不同的角度得到了认识，概括起来，主要可以分为以下几种观点：

1. 算子和非算子的观点

将动作细化作为一种算子来看待的观点类似于顺序程序设计中函数（或者过程）的调用，即在进程代数的语法中明确引入一个算子，称为动作细化算子，记为 $P[a \rightarrow P_a]$，表示将进程 P 中动作 a 的出现都替换为进程 P_a. 在这种动作细化的观点下，一个主要的研究课题就是动作细化算子的同余性（congruence）问题，即找到保持动作细化操作的等价关系. 而动作细化的非算子观点主要将动作细化作为一种实现关系来看待. 通常，一个在不同抽象层次上描述的并发系统可以看作是不同但是相互关联的系统的集合，其中每一个系统可以用一种特定的语言来描述，因此，为了关联这些不同的系统，就很有必要在这些不同的语言之间建立正确的关联关系，这种关系通常称为实现关系. 如果进程 I 比进程 S 更直接地可执行，或者根据某个语义更加确定，那么就可以认为进程 I 是进程 S 的一个实现.

动作细化的算子和非算子观点虽然都是解决同一个问题的，但是这两种方法之间却不是很容易比较的，它们各自有各自的优点，算子观点直观，容易理解，与传统的顺序系统的层次设计概念相符，但是算子观点导致一个系统只有一个实现（根据动作细化操作），因此没有实现的正确性概念，而非算子观点则允许一个系统有多个可能的实现，只要符合定义的实现关系即可. 因此两种方法各有其适合的场合.

2. 原子和非原子的观点

动作细化的原子观点认为动作是具有原子性的，因此动作细化也需要保证这种原子性，即细化以后的动作也应看成一个整体，其中的动作要么不执行，要么全部执行且不能被中断. 而动作细化的非原子观点则更自由，没有对动作的原子性限制，因此允许细化以后的动作执行被其他动作中断. 这两种观点的区别很容易用图 5.3 中的例子来加以说明，图中左边给出了 $a.0||b.0$ 的一个 LTS 表示，右边是将动作 a 细化为 $a_1.a_2.0$ 在两种不同观点下的细化结果. 可见，在原子动作细化观点下，a_1 与 a_2 之间的状态是不可见的（图中用虚线圆圈表示的状态），并且动作 b 不能在 a_1 与 a_2 的执行期间执行，这就体现了原子性的两个含义：动作执行的完整性与不可中断性.

原子动作细化的观点在某些情况下是非常适用的，例如，将一种语言用另一种

语言实现时，通常将前者的基本命令用后者的基本命令组合而成，在这种情况下，保持前者基本命令的原子性对于实现的正确性是非常重要的. 然而，动作细化的非原子观点同样有其适用的场合，并且总的来说，非原子的观点要比原子的观点为更多人所采用. 例如，在图 5.3 的例子中，如果动作 b 的发生与 a 完全独立的话，则没有理由限制 b 在 $a_1.a_2.0$ 的执行过程中不能执行，因此在这种情形下非原子的观点更加合适.

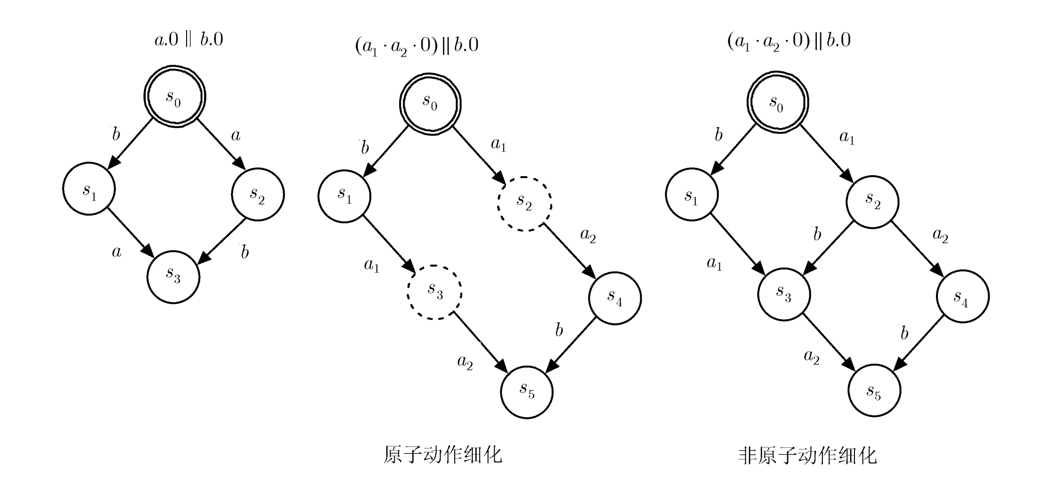


图 5.3　原子和非原子的动作细化区别

3. 语法和语义的观点

由于并发系统既可以有语言层次上的描述，又有对应的语义模型描述，因此动作细化也可以分别在语法和语义上定义，这就导致了语法动作细化和语义动作细化两种概念. 在语法动作细化方法中，动作细化的处理类似于顺序程序设计中过程调用的拷贝规则（即被调用的过程的过程体直接嵌入调用点）. 这样，动作细化就类似一个语法替换的过程. 而语义动作细化的语义观点则将这种替换操作定义在用于解释它的语义域里（例如事件结构）. 通过在并发系统的语义模型上定义动作细化操作，可以使得动作细化的含义更加直观和精确，从而避免了语法细化可能产生的抽象层次上的混淆.

5.1.3　同余性问题

在动作细化的算子观点中，一个很自然的关键问题就是找到一个等价关系，使得该等价关系关于动作细化操作（算子）是同余的. 这里，同余的概念指的是，给定一个等价关系 $\simeq$，若 $P \simeq Q$ 且 $P_1 \simeq Q_1$，则必有 $P[a \to P_1] \simeq Q[a \to Q_1]$，若只要

$P \simeq Q$, 就有 $P[a \to P_a] \simeq Q[a \to P_a]$, 则称等价关系 $\simeq$ 关于动作细化是保持的.

这里动作细化的同余性研究给出了与其他从抽象层次细化到具体层次的层次设计方法所不同的一个视角，在其他的层次设计方法中，从抽象层增加更多的实现细节转化到下一层很有可能导致底层系统的不同实现对于抽象层来说是无法区分的，而在动作细化操作下，如果能找到一个关于该操作同余的等价关系，则在最高抽象层次上等价的两个系统 P 与 Q 经过动作细化以后得到的底层实现也是保持等价的. 由于相同的动作是以等价（或相同）的方式实现（细化）的，因此这种等价关系的同余性（保持性）就保证了任何在细化以后出现的不同都已经在抽象层次上得到了反映.

同余性（或者保持性）问题是动作细化的算子观点研究中的一个中心问题，van Glabbeek R J [104] 已经详细讨论了各种功能行为等价关系（从线性时间到分支时间）在动作细化下的同余性问题，得到的结论是大多数交织语义下的等价关系关于动作细化都是不保持的，而在真并发语义下则很容易得到保持动作细化的等价关系. 通常的做法是，先采用某个已经建立好的等价关系概念，如果它关于动作细化不同余（保持），则尝试找到一个包含该等价关系的最粗的等价关系，使得它具有所期望的性质 [107,108].

解决同余性问题的另外一种做法是，给定一个已经建立好的关于动作细化不同余的等价关系，通过限制允许的细化操作或者限制所能表示的系统类别来使得在这些限制条件下的操作或者系统具有所期望的性质 [35].

5.2 基 本 假 设

如前所述，动作细化有很多不同的观点，具体采用什么观点要视具体的应用而定. 本章研究的 IMC 上的动作细化有以下的基本假设：

(1) 动作细化作为一个操作算子看待，即将动作细化定义为 IMC 上的一种操作，而不是一种实现关系. 因为算子的观点可以既简单又直观地反映出层次化的概念和特征.

(2) 动作细化是非原子的. 由于 IMC 是一种并发系统模型，动作的非原子性更适合系统的并发执行，因此对动作细化采用非原子的观点，它能更准确地反映出并发系统的执行特征.

(3) 既有语法动作细化又有语义动作细化，即将在语法和语义两个层次上考虑 IMC 的动作细化. 在语法层次上将在 IMC 的代数刻画语言 IMPA 上增加一个动作细化算子，并在语义层次上给出 IMC 模型上的动作细化操作解释.

另外，为了使得动作细化的概念更加清晰与合理，需要对 IMC 模型的基本概念作一个小小的修改，即作为系统构成基础的动作的概念需要更明确化一点，在本

章中，将动作的执行看成是需要时间的，即一个动作的执行不是瞬时完成的，需要有一段持续时间（duration）. 并且，对于动作细化操作来说，只考虑外部动作的细化操作，内部动作是不能考虑细化的. 为此，定义函数 $k : \mathrm{Act} \to R_{\geqslant 0}$ 用于给每个动作确定一个持续时间，并且约定 $k(\tau) = 0$，即内部动作的执行时间忽略不计，将其执行时间认为是 0，因此不影响外部动作的行为特征.

5.3 基于 IMC 代数刻画的语法细化

先考虑 IMC 语法层次的动作细化. 由于动作在本章中是有执行时间的，因此首先需要定义一个进程的执行时间的概念.

设 $P \in$ IMPA，根据 (3.3) 式，P 可以展开成只含前缀与选择操作的标准形式，称其中每一个选择项（$\gamma_1.\gamma_2.\cdots.\gamma_n.0$）为 P 的一个迹（trace），即 P 的一个执行路径. 由于在IMPA中存在递归表达式，因此 P 的一个执行路径当中可能包含无限的前缀操作（循环），称为无限迹（infinite trace）. 而对于不含递归项的 IMPA 表达式，很显然，它所产生的迹都是有限的，称之为非递归（recursion-free）表达式.

例 5.3.1 设

$$P = (a.(\lambda).b.0) \parallel_a (a.0 + (\mu).c.0) \ ,$$

$$Q = \mu x.(a.(\lambda).x) \ .$$

则对于进程 P，根据展开定律，有

$$\begin{aligned} P &= a.(\lambda).b.0 + (\mu).((a.(\lambda).b.0) \parallel_a c.0) \\ &= a.(\lambda).b.0 + (\mu).(a.(((\lambda).b.0) \parallel_a c.0) + c.a.(\lambda).b.0) \\ &= a.(\lambda).b.0 + (\mu).a.((\lambda).(b.0 \parallel_a c.0) + c.(\lambda).b.0) + (\mu).c.a.(\lambda).b.0 \\ &= a.(\lambda).b.0 + (\mu).a.(\lambda).b.c.0 + (\mu).a.(\lambda).c.b.0 + (\mu).a.c.(\lambda).b.0 + (\mu).c.a.(\lambda).b.0 \ . \end{aligned}$$

因此，P 共有5条有限的迹. 而对于进程 Q，它只包含一条无限的迹

$$a.(\lambda).a.(\lambda).a.(\lambda).\cdots \ .$$

以下，令 $\mathrm{tr}(P)$ 表示 P 的所有迹的集合，即

$$\mathrm{tr}(P) = \{\gamma_1.\gamma_2.\cdots.\gamma_n.0 \mid \gamma_i \in \mathrm{Act} \cup (\mathbb{R}^+)\}.$$

用 σ 来表示 $\mathrm{tr}(P)$ 中的一个元素，则 σ 代表了 P 的一个完整的执行路径（或称为最大路径），其中包括马尔可夫转移和动作转移. 对于每一个马尔可夫转移，它代表的是一个延迟时间 D_{γ_i} $(\gamma_i \in (\mathbb{R}^+))$，而每个动作又有一个执行时间 $k(\gamma_i)$ $(\gamma_i \in \mathrm{Act})$，

因此，一个迹的执行时间为

$$\mathrm{tm}(\sigma)=\sum_{\gamma_i\in(\mathbb{R}^+)} D_{\gamma_i}+\sum_{\gamma_i\in\mathrm{Act}} k(\gamma_i).$$

注意，$\mathrm{tm}(\sigma)$ 实际上是一个随机变量，它由两部分组成，一部分是马尔可夫转移造成的延迟时间总和，另一部分则是这一条迹中所有动作执行时间总和. 以下，令 $T(P)$ 表示进程 P 中所有迹的执行时间的集合，即

$$T(P)=\{\mathrm{tm}(\sigma)\mid\sigma\in\mathrm{tr}(P)\}.$$

现在，可以在 IMPA 上定义动作细化了. 根据动作细化的算子观点，通过在 IMPA 上增加一个动作细化算子来实现 IMC 的语法动作细化操作. 为此，给出以下扩充的 IMPA 定义.

定义 5.3.2 设 $a\in\mathrm{Obs}, A\subseteq\mathrm{Act}, \lambda\in\mathbb{R}^+, x\in\mathrm{Var}, P_a\in\mathrm{IMPA}$，一个**带动作细化操作的交互式马尔可夫进程代数**$\mathrm{IMPA}^{\mathrm{AF}}$ 由以下语法产生：

$$P::=0\mid a.P\mid(\lambda).P\mid P;P\mid P+P\mid P\parallel_A P\mid P\backslash A\mid P[a\rightarrow P]\mid x\mid\mu x.P.$$

与第三章介绍的 IMPA 相比，$\mathrm{IMPA}^{\mathrm{AF}}$ 在形式上只增加了一个动作细化算子 $P[a\rightarrow P_a]$. 其他算子的含义与 IMPA 相同，而对于动作细化算子，它的直观含义就是将进程 P 中所有的动作 a 都替换成进程 P_a.

与文献 [30, 80] 类似，为了避免不同抽象层次之间可能产生的混淆，必须将某些“边界”动作排除在细化操作之外. 对一个表达式 $P\in\mathrm{IMPA}^{\mathrm{AF}}$ 来说，这些动作是其中被算子 $\backslash A$ 抽象掉的动作，记为 $\mathrm{Sort}(P)$，其形式定义如下：

$$\begin{aligned}
\mathrm{Sort}(0)&=\varnothing;\\
\mathrm{Sort}(\circ P)&=\mathrm{Sort}(P),\quad\circ\in\{a.,(\lambda).\};\\
\mathrm{Sort}(P_1\circ P_2)&=\mathrm{Sort}(P_1)\cup\mathrm{Sort}(P_2),\quad\circ\in\{+,;,\parallel_A\};\\
\mathrm{Sort}(P\backslash A)&=\mathrm{Sort}(P)\cup A;\\
\mathrm{Sort}(P[a\rightarrow P_a])&=\mathrm{Sort}(P);\\
\mathrm{Sort}(\mu x.P)&=\mathrm{Sort}(P).
\end{aligned}$$

因此要求 $P[a\rightarrow P_a]$ 中被细化的动作 $a\notin\mathrm{Sort}(P)$. 同时，对用于细化动作 a 的进程 P_a，进一步要求它满足以下两个条件：

(1) P_a 是非递归的表达式，即 P_a 中不含递归项. 由于 P_a 是用于替代动作 a 的子进程，因此其执行应该是有限的，必须有明确的结束点. 这一要求是为了保证原进程的行为不被子进程的行为所破坏（如引起死锁等）.

(2) $T(P_a)=\{k(a)\}$，即 P_a 的执行时间应与动作 a 的执行时间相同，这是在动作具有执行时间的条件下的一个自然要求. 由于 P_a 的执行时间由一部分确定的动作的执行时间与另一部分随机的延迟时间组成，这一条件实际上相当于要求 $tr(P_a)$ 中每一条迹的动作执行时间总和不超过 $k(a)$.

上面的第二个条件实际上也保证了我们的动作细化不允许所谓的遗忘细化 (forgetful refinement) [104]，即将动作 a 替换成空操作，这种替换不仅与直观不符，而且会极大地改变并发系统的行为，并且这种改变不是可以由从高层次向低层次的这种转换操作解释的 [102]，因此不在我们的考虑之列.

IMPA^{AF} 中的动作细化操作的结构化操作语义由表 5.1 给出. 由表中可以看出，如果 P 中被细化的动作被执行，则 P_a 中的第一个转移 (动作转移 $\xrightarrow{a'}$ 或者马尔可夫转移 $\overset{\lambda}{\dashrightarrow}$) 将代替动作 a 被执行，同时 P_a 变为 P'_a. 如果 $P'_a\neq 0$，即动作 a 的细化进程 P_a 还没有执行完，则执行的结果得到的是进程 P 中将动作 a 替换成 P'_a 的进程；如果 $P'_a=0$，说明动作 a 的细化进程 P_a 已经执行完毕，则执行的结果得到的是原进程 P 执行动作 a 以后的结果 P' 在原动作细化算子作用下的进程. 注意到在这一过程中，动作 a 的执行被 P_a 的执行所代替，而在 P_a 执行的过程中，原进程 P 中与动作 a 无关的动作 (即并发执行的动作) 的执行是不受影响的，它们在 P_a 的执行期间可以随时执行，这就反映了我们所假设的动作细化的非原子观点.

表 5.1 IMPA^{AF} 中动作细化的结构化操作语义

$$\frac{P\xrightarrow{a}P'\wedge P_a\xrightarrow{a'}P'_a\wedge P'_a\neq 0}{P[a\to P_a]\xrightarrow{a'}P[a\to P'_a]}$$

$$\frac{P\xrightarrow{a}P'\wedge P_a\xrightarrow{a'}P'_a\wedge P'_a=0}{P[a\to P_a]\xrightarrow{a'}P'[a\to P_a]}$$

$$\frac{P\xrightarrow{a}P'\wedge P_a\overset{\lambda}{\dashrightarrow}P'_a\wedge P'_a\neq 0}{P[a\to P_a]\overset{\lambda}{\dashrightarrow}P[a\to P'_a]}$$

$$\frac{P\xrightarrow{a}P'\wedge P_a\overset{\lambda}{\dashrightarrow}P'_a\wedge P'_a=0}{P[a\to P_a]\overset{\lambda}{\dashrightarrow}P'[a\to P_a]}$$

$$\frac{P\xrightarrow{b}P'}{P[a\to P_a]\xrightarrow{b}P'[a\to P_a]}\ (b\neq a)$$

$$\frac{P\overset{\lambda}{\dashrightarrow}P'}{P[a\to P_a]\overset{\lambda}{\dashrightarrow}P'[a\to P_a]}$$

对于马尔可夫转移，由于它与动作的执行无关，很显然，它对动作细化操作的执行没有影响. IMC 对动作和延迟时间的正交结合使得我们可以对动作和延迟时间分别考虑，相互之间不会造成干扰，IMC 模型的优点在这里再一次得到了体现.

例 5.3.3 以下是IMPA^{AF} 表达式的例子：

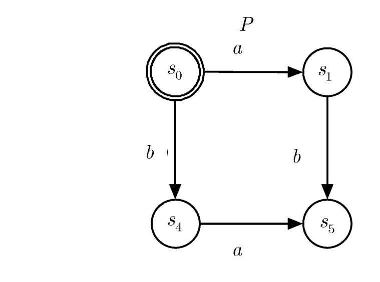
(1) $P=(a.0\,|||\,b.0)[a\to P_a]$;

(2) $P_a = a_1.(\lambda).a_2.0$,

其中，进程 P 代表两个进程的并发执行，同时将并发执行的一方 a 细化为进程 P_a，P_a 则是两个动作的顺序执行，中间加上一个参数为 λ 的指数分布时间延迟. 根据动作细化的操作语义，可以如下推导得出经过细化以后的进程 P 的表达式：

$$\begin{aligned}
P &= a_1.(a.0 \parallel\!\mid b.0)[a \to (\lambda).a_2.0] + b.(a.0[a \to a_1.(\lambda).a_2.0].) \\
&= a_1.((\lambda).(a.0 \parallel\!\mid b.0)[a \to a_2.0] + b.(a.0[a \to (\lambda).a_2.0])) + b.a_1.(\lambda).a_2.0 \\
&= a_1.(\lambda).(a_2.(a.0 \parallel\!\mid b.0)[a \to 0] + b.(a.0[a \to a_2.0])) + a_1.b.(\lambda).a_2.0 \\
&\quad +b.a_1.(\lambda).a_2.0 \\
&= a_1.(\lambda).a_2.b.0 + a_1.(\lambda).b.a_2.0 + a_1.b.(\lambda).a_2.0 + b.a_1.(\lambda).a_2.0 \ .
\end{aligned}$$

可见，经过动作细化以后的进程 P 中，动作 b 与 a 的细化进程中的动作可以任意交叉执行，保证了交织语义的真正含义. 图5.4给出了细化前与细化后的进程 P 的IMC模型.



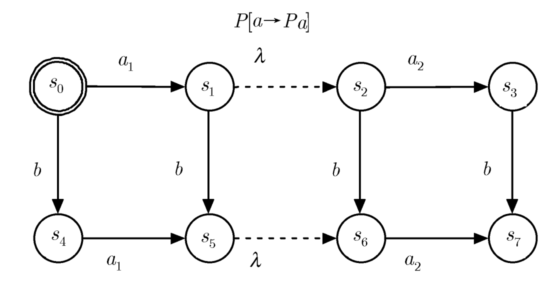


图 5.4　P 与 $P[a \to P_a]$ 的 IMC 模型

5.4　语义细化

本节将在 IMC 模型上定义动作细化操作，即语义层次上的动作细化. 由于模型的直观性，它使得我们可以更清楚地看到动作细化是如何将高层次的结构描述转换到低层次的结构描述的.

令 $\mathcal{M}$ 表示所有 IMC 模型的全体，$M=(S,\mathrm{Act},\longrightarrow,\dashrightarrow,s_0)$ 是一个 IMC，对一个给定的 $M\in\mathcal{M}$，同时也用 $S_M,\mathrm{Act}_M,\longrightarrow_M,\dashrightarrow_M,s_{0M}$ 来表示 M 的各个分量. 对于类似 S_{M_i} 的分量形式，为了简洁，记为 S_i，即只保留数字下标.

用 $\mathrm{Init}(M)$ 和 $\mathrm{Exit}(M)$ 分别表示 M 的开始状态集和结束状态集，即

$$\mathrm{Init}(M)=\{s_0\in S\},\quad \mathrm{Exit}(M)=\{s\in S\mid s\not\longrightarrow,s\not\dashrightarrow\}.$$

注意到在本书定义的 IMC 当中，$\mathrm{Init}(M)$ 实际上是个单元集（只有一个开始状态），而 $\mathrm{Exit}(M)$ 中的状态是既不能执行动作转移，也不能执行马尔可夫转移的状态（类似于 CTMC 中的吸收态），它则有可能是空集.

与语法动作细化类似，首先定义 IMC 上的迹的概念，它与第三章定义的 IMC 上的路径概念类似，都是用于描述一个 IMC 的执行情况，只是两个概念关注的重点不一样. 考虑以下两种类型的迹.

定义 5.4.1 一个序列 $\gamma=\gamma_1\gamma_2\cdots\gamma_n\in(\mathrm{Act}\cup(\mathbb{R}^+))^*$ 称为 $M\in\mathcal{M}$ 的**一个迹**（trace），如果存在 $s_0,\cdots,s_n\in S$，使得 $s_{i-1}\overset{\gamma_i}{\longmapsto}s_i(i=1,\cdots,n)$. 其中 $\longmapsto=\longrightarrow\cup\dashrightarrow$.

定义 5.4.2 一个序列 $\alpha=\alpha_1\alpha_2\cdots\alpha_n\in(\mathrm{Obs}\cup(\mathbb{R}^+))^*$ 称为 $M\in\mathcal{M}$ 的**一个可见迹**（observational trace），如果存在 $s_0,\cdots,s_n\in S$，使得 $s_{i-1}\overset{\alpha_i}{\Longmapsto}s_i(i=1,\cdots,n)$. 其中 $\Longmapsto$ 表示 $\overset{\tau}{\Longrightarrow}\overset{\alpha_i}{\longmapsto}\overset{\tau}{\Longrightarrow}$，即一个 α_i 转移，前后跟着任意（包括0）个内部动作转移.

M 中所有迹的集合记为 $\mathrm{tr}(M)$，所有可见迹的集合记为 $\mathrm{tr}_o(M)$. 在可见迹中，把一个执行路径中的内部动作都抽象掉了，因此内部动作转移不影响可见迹的集合.

对于 $\gamma\in\mathrm{tr}(M)$，它可能是一个有限或无限的序列. 当 IMC 中的转移关系存在一个回路时，即 $\exists s_k,s_k\overset{\gamma_i}{\longmapsto}s_j\overset{\gamma_j}{\longmapsto}\cdots\overset{\gamma_k}{\longmapsto}s_k$，就有可能得到一个无限的迹. 用 $\mathcal{M}^{\mathrm{inf}}$ 来表示其中的转移关系存在回路的 IMC 的集合[1]. 若 $\mathrm{tr}(M)$ 中的迹都是有限的，则 $\gamma=\gamma_1\gamma_2\cdots\gamma_n\in\mathrm{tr}(M)$ 称为最长的迹，如果 $\gamma_n\in\mathrm{Exit}(M)$，即 γ 不能再增加其长度了. 对于可见迹，也可以类似的定义最长可见迹的概念.

例 5.4.3 如图5.5所示的IMC M 中，有 $\mathrm{tr}(M)$ 由 $(1.5)\tau b(2),(1.5)\tau(0.8),a(2)$ 及其前缀组成，而 $\mathrm{tr}_o(M)$ 则由 $(1.5)b(2),(1.5)(0.8),a(2)$ 及其前缀组成.

有了 IMC 上的迹的概念，就可以定义 IMC 模型上的迹的执行时间. 与进程 P 的迹的执行时间类似，定义[2]：

$$\mathrm{tm}(\gamma)=\sum_{\gamma_i\in(\mathbb{R}^+)}D_{\gamma_i}+\sum_{\gamma_i\in\mathrm{Act}}k(\gamma_i),\quad \gamma\in\mathrm{tr}(M),$$

[1]该集合中的 IMC 实际上就与IMPA中含有递归项的表达式相对应.

[2]由于内部动作的执行时间为 0，因而这里定义的迹的执行时间与可见迹的执行时间是相同的，因此不需要对可见迹另外定义.

其中 D_{γ_i} 的含义同前，表示一个满足参数为 γ_i 的指数分布随机变量. 因此，$\mathrm{tm}(\gamma)$ 同样是一个随机变量. M 的所有迹的执行时间集合记为

$$T(M) = \{\mathrm{tm}(\gamma) \mid \gamma \in \mathrm{tr}(M)\}.$$

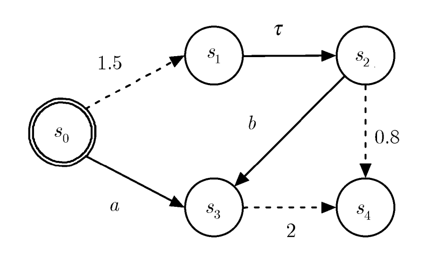


图 5.5 IMC 的迹与可见迹图例

现在，可以在 IMC 上定义动作细化操作了. 我们采用的方法与文献 [44,45,70,73,74] 中的相同，是通过一个细化函数来实现语义层次上的动作细化的.

定义 5.4.4 函数 $f: \mathrm{Obs}\backslash\mathrm{Act}_0 \to \mathcal{M}\backslash\mathcal{M}^{\mathrm{inf}}$ 称为一个**细化函数**，如果对于任意 $a \in \mathrm{Obs}$，都有 $T(f(a)) = \{k(a)\}$.

$f(a)$ 称为动作 a 的细化. 由于只考虑外部动作的细化，因此要求 a 是一个外部动作，而 Act_0 表示不需要（或者不能）被细化的动作集合. 由上面的定义可以看到，$f(a)$ 中的转移关系是不能包含有回路的，因为这有可能导致 $f(a)$ 的执行无法结束. 通过排除有回路的 IMC，就保证了 $f(a)$ 的结束状态集不为空，即 $\mathrm{Exit}(f(a)) \neq \varnothing$. 此外，与语法细化中的要求一样，还要求 $f(a)$ 的所有迹的执行时间与动作 a 的持续时间相等，这同样也是在动作 a 有执行时间的假设下的一个合理要求.

例 5.4.5 如图5.6所示的两个IMC，设 M 中动作 a 的执行时间为5，M_a 中 a_1 和 a_2 的执行时间分别为5和3，有

$$T(M_a) = \{5, D_\mu + 3\}.$$

因此当 $D_\mu + 3$ 等于5时，有 $T(M_a)$ 等于 5. 设 $\mathrm{Obs}\backslash\mathrm{Act}_0 = \{a\}$，定义 $f(a) = M_a$，则 f 是一个细化函数，$f(a)$ 就是 a 的一个细化.

现在的问题是，给定一个细化函数 f，怎么将其应用到IMC上，得到一个细化后的IMC. 设 $\mathrm{Pre}_M(a)$ 和 $\mathrm{Post}_M(a)$ 分别表示 $M \in \mathcal{M}$ 中的 a 动作转移的前趋状态集和后继状态集，即

$$\mathrm{Pre}_M(a) = \{s \in S_M \mid \exists s' \in S_M, s \xrightarrow{a} s'\},$$

$$\mathrm{Post}_M(a) = \{s \in S_M \mid \exists s' \in S_M, s' \xrightarrow{a} s\},$$

并且记 $\text{Pre}_M(A)=\bigcup_{a\in A}\text{Pre}(a), \text{Post}_M(A)=\bigcup_{a\in A}\text{Post}(A)\ (A\subseteq \text{Act}_M)$，则定义IMC的细化如下：

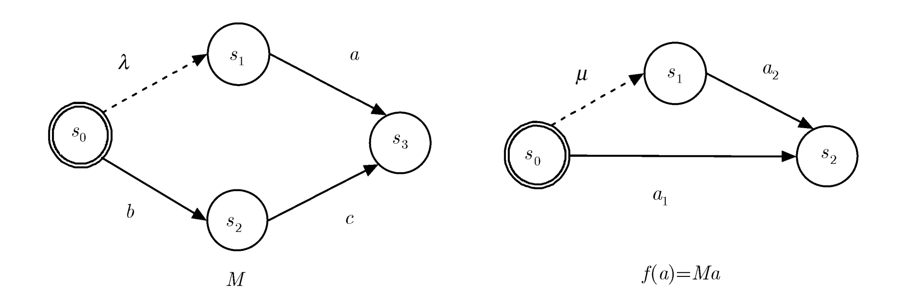


图 5.6 动作细化函数示例

定义 5.4.6 设 $M\in\mathcal{M}$，f 是一个细化函数，则 **M 的细化**定义为

$$f(M)=(S_f,\text{Act}_f,\longrightarrow_f,\dashrightarrow_f,s_{0f}),$$

其中

- 状态集

$$S_f=S_M\cup(\bigcup_{a\in\text{Obs}\backslash\text{Act}_0}S_{f(a)});$$

- 动作集

$$\text{Act}_f=(\text{Act}_M\backslash(\text{Obs}\backslash\text{Act}_0))\cup(\bigcup_{a\in\text{Obs}\backslash\text{Act}_0}\text{Act}_{f(a)});$$

- 动作转移关系

$$\begin{aligned}\longrightarrow_f\ =\ &\longrightarrow_M\backslash(\text{Pre}_M(\text{Obs}\backslash\text{Act}_0)\times(\text{Obs}\backslash\text{Act}_0)\times\text{Post}_M(\text{Obs}\backslash\text{Act}_0))\cup\\&(\text{Pre}_M(\text{Obs}\backslash\text{Act}_0)\times\{\tau\}\times(\bigcup_{a\in\text{Obs}\backslash\text{Act}_0}\text{Init}(f(a))))\cup\\&((\bigcup_{a\in\text{Obs}\backslash\text{Act}_0}\text{Exit}(f(a)))\times\{\tau\}\times\text{Post}_M(\text{Obs}\backslash\text{Act}_0))\cup\\&(\bigcup_{a\in\text{Obs}\backslash\text{Act}_0}\longrightarrow_{f(a)});\end{aligned}$$

- 马尔可夫转移关系

$$\dashrightarrow_f=\dashrightarrow_M\cup(\bigcup_{a\in\text{Obs}\backslash\text{Act}_0}\dashrightarrow_{f(a)});$$

- 初始状态

$$s_{0f}=s_{0M}.$$

由以上定义可知，一个 IMC 的细化仍然是一个 IMC, 其中细化后的 IMC 的状态集合是原 IMC 的状态集与所有用于细化的 IMC 的状态集的并集. 细化后的动作集是原 IMC 动作集合中除了被细化的动作集合之外的集合与所有用于细化的 IMC 的动作集的并集. 对于细化后的动作转移关系与马尔可夫转移关系，可以用图 5.7 来直观的解释.

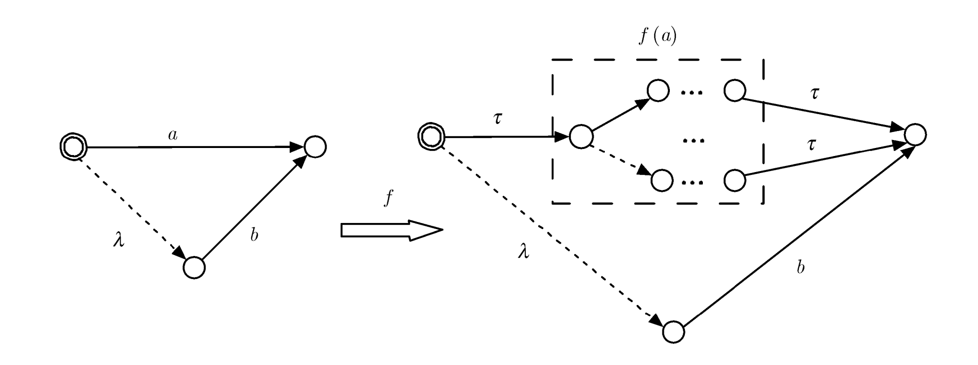


图 5.7 IMC 的动作细化示意图

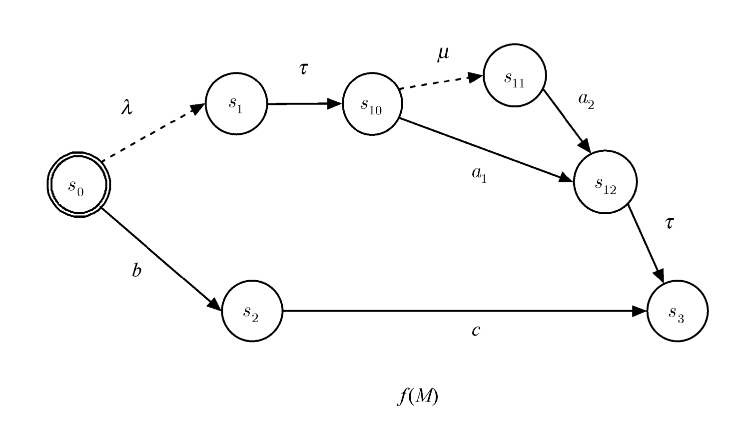


图 5.8 例 5.4.5 中 M 的细化结果

由图可知, IMC 中被细化的动作 a 的转移关系被 $f(a)$ 所替代，其中在 a 的前趋状态与 $f(a)$ 的开始状态之间和 $f(a)$ 的结束状态与 a 的后继状态之间分别增加了一个内部动作转移，用以连接原系统与用于细化的子系统. 由于内部动作的不可见性，增加的两组内部动作转移既不影响原系统的状态，也不影响原系统的行为. 因此，细化后的动作转移就是原系统中的动作转移去掉被细化的动作转移，再加上用于细化的 IMC 的动作转移和两组新增的用于连接细化 IMC 的开始与结束集合的内部动作转移. 由于动作细化只涉及动作转移关系，因此马尔可夫转移关系不受影

响，所以细化后的马尔可夫转移关系就是原 IMC 的马尔可夫转移关系与用于细化的 IMC 的马尔可夫转移关系的并集. 最后，细化后的 IMC 的初始状态显然还是原 IMC 的初始状态.

例 5.4.7 继续考虑例 5.4.5 中的IMC，$f(a)$ 是动作 a 的一个细化，将 f 应用到 M 上，则可以得到图5.8所示的 M 的细化.

5.5 性　质

以上分别在 IMC 的语法 (IMPA) 和语义 (模型) 层次上定义了动作细化操作，本节将研究我们定义的动作细化操作所具有的性质. 与通常的做法一样，我们主要研究动作细化算子的同余性问题和语法语义两种细化的一致性问题. 这两个问题都是在一定的等价关系下讨论的，本章主要涉及两种等价关系，一种是线性时间的迹等价，另一种是分支时间的互模拟等价.

5.5.1 交织语义的等价关系概念

在本章中，等价关系主要用于给 IMPA 提供语义基础，即建立进程间的相等关系. 在语义框架下，等价关系主要是用于比较两个不同进程的行为，因此通常是定义在两个进程之间的.

在第四章，已经在 IMC 上定义了各种分支时间的等价关系，包括强互模拟和弱互模拟等价关系. 其中对动作转移与马尔可夫转移是分别考虑的，对于马尔可夫转移所满足的条件，它只反映系统的随机特性，并不反映系统的行为特征，因此不是本章关注的重点，本章重点关注的是等价关系的行为描述方面，也就是与动作转移相关的等价概念.

为了统一 IMC 中两种转移关系所满足的条件，同时又能表达两个系统行为比较的特征，本章采用以下修改的互模拟等价的概念.

定义 5.5.1 设 $M_1, M_2 \in \mathcal{M}$，一个定义在 $S_1 \times S_2$ 上的关系 R 称为 M_1 **和 M_2 之间的一个强交织互模拟等价关系** (strong interleaving bisimulation)，当且仅当 $(s_{01}, s_{02}) \in R$，且若 $(s_1, s_2) \in R$，则

(1) $s_1 \overset{\gamma_i}{\longmapsto}_1 s_1', \gamma_i \in \mathrm{Act} \cup (\mathbb{R}^+) \ \Rightarrow\ \exists s_2' \in S_2, s_2 \overset{\gamma_i}{\longmapsto}_2 s_2'$ 并且$(s_1', s_2') \in R$;

(2) $s_2 \overset{\gamma_i}{\longmapsto}_2 s_2', \gamma_i \in \mathrm{Act} \cup (\mathbb{R}^+) \ \Rightarrow\ \exists s_1' \in S_1, s_1 \overset{\gamma_i}{\longmapsto}_1 s_1'$ 并且$(s_1', s_2') \in R$.

如果 M_1 和 M_2 之间存在一个交织互模拟等价关系 R，则称 M_1 和 M_2 是强交织互模拟的，记为 $M_1 \sim_{\mathrm{ib}} M_2$.

类似的，如果将内部动作 τ 抽象掉，就可以得到以下弱交织互模拟等价的概念.

定义 5.5.2 设 $M_1, M_2 \in \mathcal{M}$，一个定义在 $S_1 \times S_2$ 上的关系 $\widetilde{R}$ 称为 M_1 **和 M_2 之间的一个弱交织互模拟等价关系** (weak interleaving bisimulation)，当且仅当 $(s_{01}, s_{02}) \in \widetilde{R}$，且若 $(s_1, s_2) \in \widetilde{R}$，则

(1) $s_1 \overset{\alpha_i}{\Longmapsto}_1 s_1', \alpha_i \in \mathrm{Obs} \cup (\mathbb{R}^+) \;\Rightarrow\; \exists s_2' \in S_2, s_2 \overset{\alpha_i}{\Longmapsto}_2 s_2'$ 并且$(s_1', s_2') \in \widetilde{R}$;

(2) $s_2 \overset{\alpha_i}{\Longmapsto}_2 s_2', \alpha_i \in \mathrm{Obs} \cup (\mathbb{R}^+) \;\Rightarrow\; \exists s_1' \in S_1, s_1 \overset{\alpha_i}{\Longmapsto}_1 s_1'$ 并且$(s_1', s_2') \in \widetilde{R}$.

如果 M_1 和 M_2 之间存在一个弱交织互模拟等价关系 $\widetilde{R}$, 则称 M_1 和 M_2 是弱交织互模拟的，记为 $M_1 \approx_{\mathrm{ib}} M_2$.

在上面定义的交织互模拟等价关系中，不再明确区分动作和马尔可夫转移，将马尔可夫转移上的时间延迟参数看成是一个特殊的动作，统一按照动作转移满足的条件来定义. 这是由于 IMPA 操作语义中的马尔可夫转移与动作转移是同样交织执行的，因此它们在本质上没有什么区别 (除了马尔可夫转移不能同步之外) .

虽然定义 5.5.1 和定义 5.5.2 中的等价关系是定义在两个 IMC 之间的，但仍然是定义在状态集合上的，由于在交织互模拟等价关系中每个马尔可夫转移都需要有一个相同的转移来匹配，显然也满足第四章定义的互模拟等价关系中等价状态到相同等价类中的总转移率相等的条件，因此，这样定义的等价关系可以看作是第四章相应的互模拟等价关系的一个特例.

下面，再给出 IMC 上的交织迹等价的概念.

定义 5.5.3 设 $M_1, M_2 \in \mathcal{M}$，若 $\mathrm{tr}(M_1) = \mathrm{tr}(M_2)$，则称 M_1 和 M_2 是**强交织迹等价**(strong interleaving trace equivalence) 的，记为 $M_1 \sim_{\mathrm{it}} M_2$.

定义 5.5.4 设 $M_1, M_2 \in \mathcal{M}$，若 $\mathrm{tr}_o(M_1) = \mathrm{tr}_o(M_2)$，则称 M_1 和 M_2 是**弱交织迹等价**(weak interleaving trace equivalence) 的，记为 $M_1 \approx_{\mathrm{it}} M_2$.

迹等价只考虑系统的执行路径，并且把具有相同执行路径集合的两个系统看成是一样的，因此是一种线性时间的等价关系.

例 5.5.5 图5.9显示了两个分别与图5.6中的两个IMC等价的IMC，其中 $M' \sim_{\mathrm{eq}} M, M_a \sim_{\mathrm{eq}} M_a'$ $(\mathrm{eq} \in \{\mathrm{it}, \mathrm{ib}\})$，因为它们分别都有相同的迹的集合和相同的分支结构.

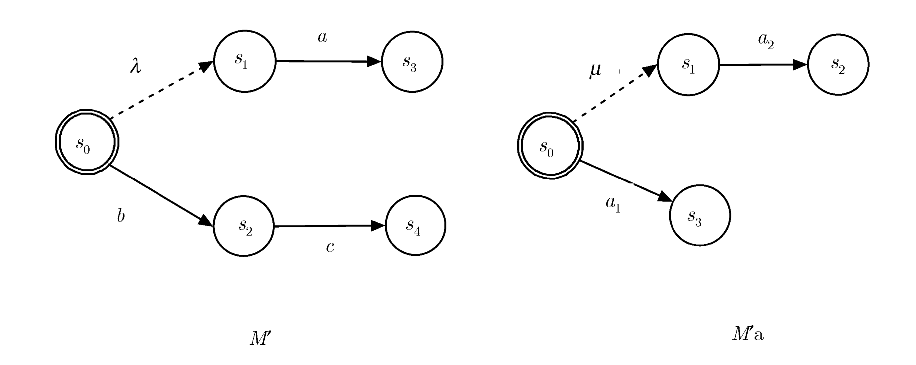


图 5.9 分别与图 5.6 中两个 IMC 等价的 IMC

显然，交织互模拟的两个 IMC 必然具有相同的迹，并且强等价蕴涵弱等价，因此有以下命题成立：

命题 5.5.6　对任意 $M_1, M_2 \in \mathcal{M}$，有

(1) $M_1 \sim_{\text{ib}} M_2 \ \Rightarrow\ M_1 \sim_{\text{it}} M_2$,

(2) $M_1 \approx_{\text{ib}} M_2 \ \Rightarrow\ M_1 \approx_{\text{it}} M_2$,

(3) $M_1 \sim_{\text{ib}} M_2 \ \Rightarrow\ M_1 \approx_{\text{ib}} M_2$,

(4) $M_1 \sim_{\text{it}} M_2 \ \Rightarrow\ M_1 \approx_{\text{it}} M_2$.

5.5.2　同余性

由于模型的直观性，先讨论 IMC 上的动作细化的同余性问题，也就是语义动作细化的同余性问题.

设 $M \in \mathcal{M}, \gamma = \gamma_0\gamma_1 \cdots \gamma_n \in \text{tr}(M)$，$f$ 是一个细化函数. 令 $\gamma^a \in \text{tr}(f(a))$ 并且满足它是 $\text{tr}(f(a))$ 中最长的迹，定义 γ 的一个细化如下:

$$f(\gamma, \cup_a \gamma^a) = f(\gamma_1) f(\gamma_2) \cdots f(\gamma_n),$$

其中

$$f(\gamma_i) = \begin{cases} \gamma_i, & \gamma_i \in (\mathbb{R}^+) \cup \text{Act}_0, \\ \tau\gamma^a\tau, & \gamma_i = a \in \text{Obs}\backslash\text{Act}_0. \end{cases}$$

$f(\gamma, \cup_a \gamma^a)$ 实际上是通过将 γ 中每个被细化的动作用细化后的最长迹来代替得到的. 在替换后的最长迹前后加入两个内部动作是为了与 M 的细化函数相对应. 对于马尔可夫转移的延迟参数和不能被细化的动作，在细化后的迹中仍然保持不变.

下面的引理说明，$f(M)$ 的迹由 M 的迹的细化所构成.

引理 5.5.7　设 $M \in \mathcal{M}$，f 是一个细化函数，则 $\text{tr}(f(M)) = \{f(\gamma, \cup_a \gamma^a) \mid \gamma \in \text{tr}(M), a \in \text{Obs}\backslash\text{Act}_0\}$.

证明　由 $f(\gamma, \cup_a \gamma^a)$ 的定义，它显然是 $f(M)$ 的一条执行路径，因此有 $f(\gamma, \cup_a \gamma^a) \in \text{tr}(f(M))$. 反过来，任给 $\gamma_f \in \text{tr}(f(M))$，根据 $f(M)$ 的定义，γ_f 都可以表示成以下的形式

$$\gamma_1 \cdots \gamma_i \tau \gamma^{a_i} \tau \gamma_j \cdots \gamma_n,$$

其中 $\gamma^{a_i} \in \text{tr}(f(a_i))$ 且是其中最长的迹, $a_i \in \text{Obs}\backslash\text{Act}_0$. 注意在 $f(M)$ 的定义中，我们在 $\text{Pre}_M(a_i)$ 与 $\text{Init}(f(a_i))$ 之间及 $\text{Exit}(f(a_i))$ 与 $\text{Post}_M(a_i)$ 之间引入了内部动作转移，因此在 γ^{a_i} 的前后都有一个内部动作 τ. 由此可见，γ_f 是由 M 中某一个迹将其中被细化的动作用细化后的最长迹来代替得到的. 因此 $\gamma_f \in \{f(\gamma, \cup_a \gamma^a) \mid \gamma \in \text{tr}(M), a \in \text{Obs}\backslash\text{Act}_0\}$. 因此引理结论成立.

定理 5.5.8　设 $M_1, M_2 \in \mathcal{M}$，f_1 和 f_2 是两个细化函数. 如果 $M_1 \sim_{\text{eq}} M_2$ 且对任何 $a \in \text{Obs}\backslash\text{Act}_0$，有 $f_1(a) \sim_{\text{eq}} f_2(a)$，则 $f_1(M_1) \sim_{\text{eq}} f_2(M_2)$, 其中 $\text{eq} \in \{\text{it}, \text{ib}\}$.

证明 (i) 先证 eq = it 的情况.

由引理 5.5.7 可知,

$$\mathrm{tr}(f_1(M_1)) = \{f_1(\gamma, \cup_a \gamma^a) \mid \gamma \in \mathrm{tr}(M_1), a \in \mathrm{Obs}\backslash\mathrm{Act}_0\},$$

$$\mathrm{tr}(f_2(M_2)) = \{f_2(\gamma, \cup_a \gamma^a) \mid \gamma \in \mathrm{tr}(M_2), a \in \mathrm{Obs}\backslash\mathrm{Act}_0\}.$$

又因为 $M_1 \sim_{\mathrm{it}} M_2, f_1(a) \sim_{\mathrm{it}} f_2(a)$, 有 $\mathrm{tr}(M_1) = \mathrm{tr}(M_2)$, 且对任意 $a \in \mathrm{Obs}\backslash\mathrm{Act}_0$, $\mathrm{tr}(f_1(a)) = \mathrm{tr}(f_2(a))$, 因此 $f_1(\gamma, \cup_a \gamma^a) = f_2(\gamma, \cup_a \gamma^a)$, 即

$$\mathrm{tr}(f_1(M_1)) = \mathrm{tr}(f_2(M_2)).$$

所以 $f_1(M_1) \sim_{\mathrm{it}} f_2(M_2)$.

(ii) 接下来证明 eq = ib 的情况.

由已知, $M_1 \sim_{\mathrm{ib}} M_2$ 且对任意 $a \in \mathrm{Obs}\backslash\mathrm{Act}_0, f_1(a) \sim_{\mathrm{ib}} f_2(a)$, 因此, 存在 M_1 和 M_2 之间的一个强交织互模拟等价关系 R 及 $f_1(a)$ 和 $f_2(a)$ 之间的一个强交织互模拟等价关系 R_a, 使得 $(s_{M_1}, s_{M_2}) \in R, (s_{f_1(a)}, s_{f_2(a)}) \in R_a$. 现在, 构造一个 $S_{f_1(M_1)} \times S_{f_2(M_2)}$ 上的关系 R_f 如下:

$(s_{f_1(M_1)}, s_{f_2(M_2)}) \in R_f$ 当且仅当

(1) $s_{f_i(M_i)} = s_{M_i} \in S_i$ $(i = 1, 2)$ 时, $(s_{M_1}, s_{M_2}) \in R$;

(2) $s_{f_i(M_i)} = s_{f_i(a)} \in S_{f_i(a)}$ $(i = 1, 2)$ 时, $(s_{f_1(a)}, s_{f_2(a)}) \in R_a$.

需要证明, R_f 是 $f_1(M_1)$ 和 $f_2(M_2)$ 之间的一个强交织互模拟等价关系. 首先, 有

$$(s_{0\,f_1(M_1)}, s_{0\,f_2(M_2)}) = (s_{0\,M_1}, s_{0\,M_2}) \in R.$$

因此 $(s_{0\,f_1(M_1)}, s_{0\,f_2(M_2)}) \in R_f$.

设 $(s_{f_1(M_1)}, s_{f_2(M_2)}) \in R_f$, 并且 $s_{f_1(M_1)} \stackrel{\gamma_i}{\longmapsto}_1 s'_{f_1(M_1)}$, 需要证明存在 $s'_{f_2(M_2)} \in S_{f_2(M_2)}$, 使得 $s_{f_2(M_2)} \stackrel{\gamma_i}{\longmapsto}_2 s'_{f_2(M_2)}$, 并且满足 $(s'_{f_1(M_1)}, s'_{f_2(M_2)}) \in R_f$. 分以下几种情况讨论:

(1) $s_{f_1(M_1)} \in S_1\backslash\mathrm{Pre}_{M_1}(\mathrm{Obs}\backslash\mathrm{Act}_0)$, 即 $s_{f_1(M_1)}$ 是 M_1 中除了被细化的动作的前趋状态集合之外的状态, 因此 $s'_{f_1(M_1)} \in M_1$. 根据 R_f 的定义, 可以在 M_2 中找到一个 $s'_{M_2} = s'_{f_2(M_2)}$, 使得 $s_{f_2(M_2)} \stackrel{\gamma_i}{\longmapsto}_2 s'_{f_2(M_2)}$, 并且满足 $(s'_{f_1(M_1)}, s'_{f_2(M_2)}) \in R_f$.

(2)$s_{f_1(M_1)} \in S_{f_1(a)}\backslash\mathrm{Exit}(f_1(a))$, 即 $s_{f_1(M_1)}$ 是 $f_1(a)$ 中除了退出状态集合的状态, 同理, 根据 R_f 的定义, 可以在 $f_2(a)$ 中找到一个 $s'_{M_2} = s'_{f_2(M_2)}$, 使得 $s_{f_2(M_2)} \stackrel{\gamma_i}{\longmapsto}_2 s'_{f_2(M_2)}$, 并且满足 $(s'_{f_1(M_1)}, s'_{f_2(M_2)}) \in R_f$.

(3) $s_{f_1(M_1)} \in \mathrm{Pre}_{M_1}(\mathrm{Obs}\backslash\mathrm{Act}_0)$, 即 $s_{f_1(M_1)}$ 是 M_1 中被细化动作的前趋状态集合, 则 $s'_{f_1(M_1)} = s_{0\,f_1(a)} \in S_{f_1(a)}$, 由定义, $s_{f_1(M_1)} \stackrel{\tau}{\longmapsto}_1 s_{0\,f_1(a)}$, 取 $s'_{f_2(M_2)} =$

$s_{0f_2(a)} \in S_{f_2(a)}$，则 $s_{f_2(M_2)} \overset{\tau}{\longmapsto}_2 s_{0f_2(a)}$ 且有 $(s_{0f_1(a)}, s_{0f_2(a)}) \in R_a$，因此同样有 $({s'}_{f_1(M_1)}, {s'}_{f_2(M_2)}) \in R_f$.

(4) $s_{f_1(M_1)} \in \mathrm{Exit}(f_1(a))$，即 $s_{f_1(M_1)}$ 是 $f_1(a)$ 中的退出状态，则 ${s'}_{f_1(M_1)} \in S_1$，同样，根据 M_1 的细化定义，有 $s_{f_1(M_1)} \overset{\tau}{\longmapsto}_1 {s'}_{f_1(M_1)}$，取 ${s'}_{f_2(M_2)} \in \mathrm{Post}_{M_2}(a)$，则 $s_{f_2(M_2)} \overset{\tau}{\longmapsto}_2 {s'}_{f_2(M_2)}$ 且有 $({s'}_{f_1(M_1)}, {s'}_{f_2(M_2)}) \in R$，从而 $({s'}_{f_1(M_1)}, {s'}_{f_2(M_2)}) \in R_f$.

由对称性，强交织互模拟条件的另一部分可以类似证明. 因此 R_f 是 $f_1(M_1)$ 和 $f_2(M_2)$ 之间的一个强交织互模拟等价关系，即 $f_1(M_1) \sim_{\mathrm{ib}} f_2(M_2)$.

定理 5.5.8 说明，交织迹等价和交织互模拟等价关系在我们定义的 IMC 上的动作细化操作下都是同余的.

例 5.5.9 结合例 5.4.5、例 5.4.7 和例 5.5.5，我们给出一个语义动作细化的同余性例子. 在图5.9中，设 $f'(a) = M'_a$ 是 M' 中 a 的一个细化，则有

$$M \sim_{\mathrm{eq}} M', f(a) \sim_{\mathrm{eq}} f'(a)\ (\mathrm{eq} \in \{\mathrm{it}, \mathrm{ib}\}).$$

$f(M)$ 已经在图5.8中给出，$f'(M')$ 则由图5.10给出.

很容易验证，$f(M) \sim_{\mathrm{eq}} f'(M')$.

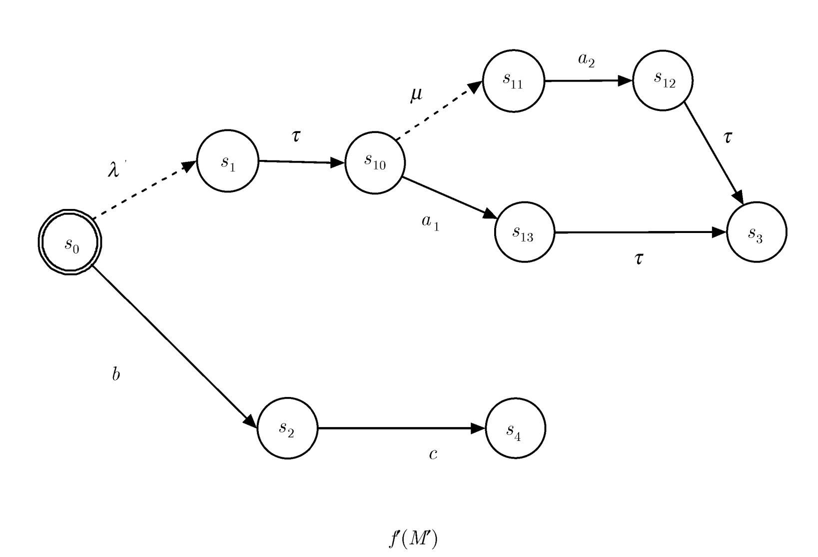


图 5.10　M' 的细化 $f'(M')$

下面再来看语法层次上的动作细化的同余性问题. 根据 $\mathrm{IMPA}^{\mathrm{AF}}$ 的操作语义，每一个 $\mathrm{IMPA}^{\mathrm{AF}}$ 表达式都可以对应得到一个 IMC 模型. 以下，为了叙述方便，用 $s : \mathrm{IMPA}^{\mathrm{AF}} \to \mathcal{M}$ 来表示这样一个操作语义映射，即 $s(P)$ 表示表达式 P 对应的 IMC 模型. 由操作语义的规则可知，$\mathrm{IMPA}^{\mathrm{AF}}$ 表达式中的动作前缀操作 $(a.)$、延迟前缀操作 $((\lambda).)$ 和选择操作 $(+)$ 分别对应于 IMC 模型的动作转移、延迟转移和分

支转移, 因此可以将前缀操作、选择操作和顺序操作对应地定义到 IMC 模型上, 即有

$$
\begin{aligned}
s(a.P) &= a.s(P), \\
s((\lambda).P) &= (\lambda).s(P), \\
s(P_1 + P_2) &= s(P_1) + s(P_2), \\
s(P_1; P_2) &= s(P_1); s(P_2).
\end{aligned}
$$

如图 5.11 所示，其中选择操作与顺序操作执行的转移是动作转移还是延迟转移依表达式 P_1（或 P_2）的第一个操作决定.

下面的例子说明，在交织语义下，迹等价和互模拟等价关系在语法动作细化下都是不保持的，因此也是不同余的.

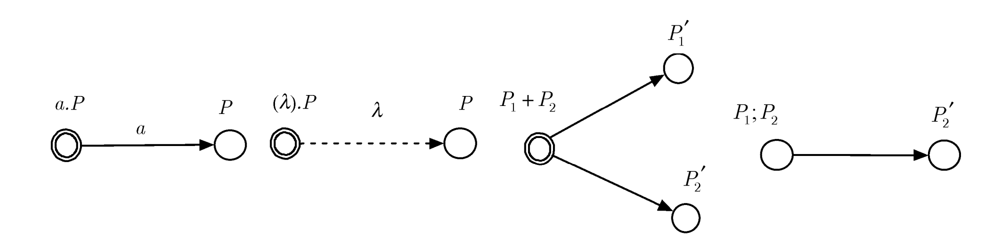


图 5.11　IMC 上的各种组合操作示意图

例 5.5.10　设

$$
\begin{aligned}
P &= a.(\lambda).0 \,|||\, (\mu).b.0\ , \\
Q &= a.(\lambda).(\mu).b.0 + a.(\mu).(\lambda).b.0 + a.(\mu).b.(\lambda).0 \\
&\quad +(\mu).b.a.(\lambda).0 + (\mu).a.b.(\lambda).0 + (\mu).a.(\lambda).b.0\ .
\end{aligned}
$$

根据操作语义，可以得到 P, Q 对应的IMC模型都为图5.12所示的IMC. 因此有 $s(P) \approx_{\mathrm{it}} s(Q)$ 和 $s(P) \approx_{\mathrm{ib}} s(Q)$.

然而，当把动作 a 细化为 $P_a = a_1.a_2.0$ 时，$s(P[a \to P_a])$ 和 $s(Q[a \to P_a])$ 分别如图5.13和图5.14所示，很显然，它们既不是交织迹等价的，也不是交织互模拟等价的. 因为在 $s(P[a \to P_a])$ 中，有 $a_1.(\mu).a2 \in \mathrm{tr}(s(P[a \to P_a])$，而在 $s(Q[a \to P_a])$ 中，有 $a_1.(\mu).a2 \notin \mathrm{tr}(s(Q[a \to P_a])$，即 $\mathrm{tr}(s(P[a \to P_a]) \neq \mathrm{tr}(s(Q[a \to P_a])$，因此 $s(P[a \to P_a]) \not\approx_{\mathrm{it}} s(Q[a \to P_a])$，进而有 $s(P[a \to P_a]) \not\approx_{\mathrm{ib}} s(Q[a \to P_a])$.

由上例可以看出，导致语法动作细化不能保持交织语义等价关系的原因在于并发算子的作用. 在交织语义下，两个并发执行的进程是可以任意交叉执行的，因此，

对并发执行的动作进行细化后，所得到的进程是在细化之后的进程之间任意交叉执行，这就与没有并发执行动作的进程的细化产生了不同.

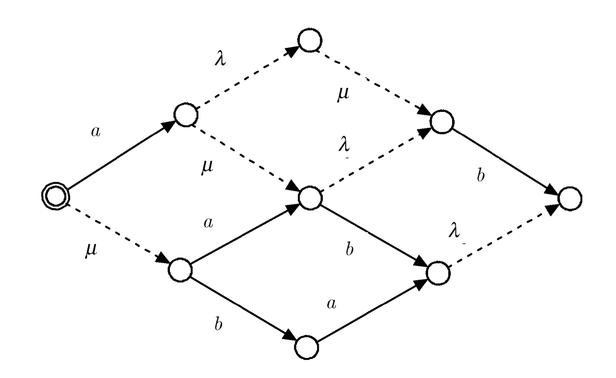


图5.12　P, Q 的操作语义模型 $s(P), s(Q)$

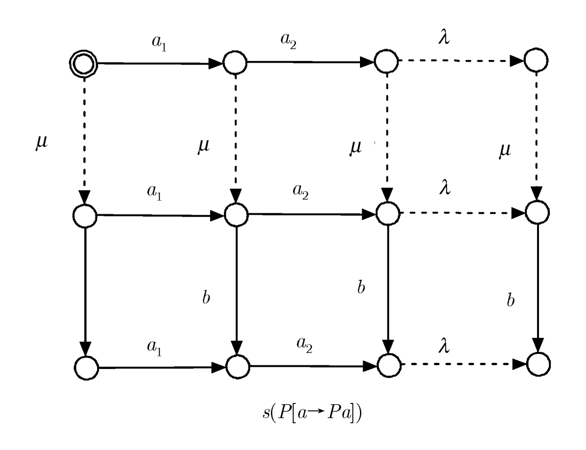


图 5.13　$P[a \to P_a]$ 的操作语义模型

为了使得语法上的动作细化能够具有我们期望的同余性质，需要对动作细化算子做一定的限制，即不允许对并发执行的动作做细化. 称这样的细化为安全的细化 (safe refinement）.

设一个表达式 P 中所有动作的集合, 用 A_P 来表示，令

$$C(P) = \{a \in A_{P_1} \cup A_{P_2} \mid P_1, P_2 \text{ 是 } P \text{ 中所有并发算子的两个子表达式，即} P_1 \parallel_A P_2\},$$

即 $C(P)$ 是表达式 P 中所有受并发算子作用子表达式的动作集合，称为关键动作集 (critical action set）.

定义 5.5.11　IMPA^{AF} 中的动作细化算子 $P[a \to P_a]$ 称为**安全的细化**，如果 $a \notin C(P)$. 安全的动作细化算子记为 $P[\bar{a} \to P_a]$.

例 5.5.12 设 $P = a.(2)b.0+(b.c.0+(1).c.0) \parallel_c c.(3).0$, 则 $C(P) = \{b, c\}$, $P[\bar{a} \to a_1.a_2.0]$ 是一个安全的细化, $P[b \to b_1.b_2.0]$, 则不是安全的细化.

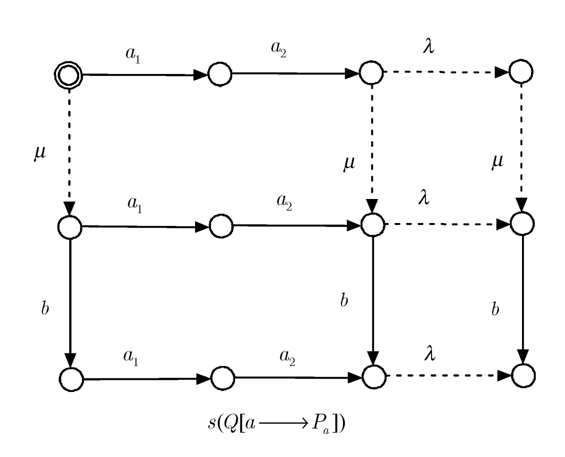


图 5.14 $Q[a \to P_a]$ 的操作语义模型

安全的细化通过限制可以被细化的动作集合避免了并发算子对动作细化造成的影响，然而这个限制却是相当苛刻的，它使得动作细化的作用范围大大缩小. 在交织语义下，要使得交织迹等价关于动作细化是保持的，要找到比这个限制更宽的条件似乎是不可能的 [35].

下面的定理说明，交织迹等价和交织互模拟等价在安全的（语法）细化下都是同余的.

定理 5.5.13 设 $P, Q, P_a, Q_a \in \text{IMPA}$，$a \in \text{Obs}\backslash\text{Act}_0$，若 $P \sim_{\text{eq}} Q, P_a \sim_{\text{eq}} Q_a$，则 $P[\bar{a} \to P_a] \sim_{\text{eq}} Q[\bar{a} \to Q_a]$，其中 $\text{eq} \in \{\text{it}, \text{ib}\}$.

证明 $P \sim_{\text{eq}} Q, P_a \sim_{\text{eq}} Q_a$ 即 $s(P) \sim_{\text{eq}} s(Q), s(P_a) \sim_{\text{eq}} s(Q_a)$，由于安全的动作细化不涉及并发算子，即相互可以并发执行的动作不允许被细化，因此对任意 $a \in (\text{Obs}\backslash\text{Act}_0) \cap C(P)$，由 $P[\bar{a} \to P_a]$ 的操作语义，P 中任何 a 的发生都被 P_a 的执行所代替，且 P 中没有与 a 并发执行的动作，因此 P_a 的执行是完整的，同理，在 $Q[\bar{a} \to Q_a]$ 中，Q_a 的执行也是完整的. 由 $s(P) \sim_{\text{eq}} s(Q), s(P_a) \sim_{\text{eq}} s(Q_a)$ 及交织迹等价和交织互模拟等价的定义可知 $s(P[\bar{a} \to P_a]) \sim_{\text{eq}} s(Q[\bar{a} \to Q_a])$，即 $P[\bar{a} \to P_a] \sim_{\text{eq}} Q[\bar{a} \to Q_a]$.

5.5.3 语法和语义细化的一致性

由以上讨论可以看出，在交织迹等价和交织互模拟等价的意义下，本章前面定义的语法和语义上的细化关于 IMPA^{AF} 的操作语义并不一致. 其原因主要在于在语法层次上，明确区分了并发算子和前缀算子与选择算子的组合，即在形式上既有 $a.0 \parallel\!| b.0$，又有 $a.b.0 + b.a.0$，而在语义层次上，这两种表达式只有一种操作语义，因

此对应的是同一个模型. 已经看到，并发算子是使得动作细化在语法和语义层次产生不一致的主要原因，所以若要保持动作细化在语法和语义层次的一致性，同样必须将并发算子排除在外, 即只能考虑安全的动作细化的一致性问题.

于是可以得到以下语法和语义细化的一致性定理.

定理 5.5.14　设 $P, P_a \in \text{IPA}$，$f : \text{Obs}\backslash\text{Act}_0 \to \mathcal{M}\backslash\mathcal{M}^{\text{inf}}$ 是一个细化函数，且对任意 $a \notin C(P), f(a) = s(P_a)$，则 $f(s(P)) \approx_{\text{eq}} s(P[\bar{a} \to P_a])$，其中 $\text{eq} \in \{\text{it}, \text{ib}\}$.

证明　对 P 的结构做归纳证明. 由于 $a \notin C(P)$，因此不需要考虑并发算子 $\|_A$. 又由语法细化的限制条件 $a \notin \text{Sort}(P)$，也不需要考虑隐藏算子 $\backslash A$.

(i) 对于 $P = 0$，结论显然成立.

(ii) 假设结论对 P_1, P_2 都成立，即

$$f(s(P_1)) \approx_{\text{eq}} s(P_1[\bar{a} \to P_a]),\ f(s(P_2)) \approx_{\text{eq}} s(P_2[\bar{a} \to P_a]),$$

则

- 对 $P = a.P_1$ 且 $a \notin \text{Act}_0$，由 $a.P_1$ 的操作语义可知，$s(a.P_1)$ 的第一个转移是 a 动作转移，然后到达 $s(P_1)$ 的第一个状态，即 $s(a.P_1) = a.s(P_1)$，因此

  $$f(s(a.P_1)) = \tau.f(a); \quad \tau.f(s(P_1)) \approx_{\text{eq}} f(a); \quad s(P_1[\bar{a} \to P_a]) = s(a.P_1[\bar{a} \to P_a]).$$

  对 $a \in \text{Act}_0$ 的情形可类似得证.
- 对 $P = (\lambda).P_1$，由于延迟前缀与动作细化无关，因此与 $P = a.P_1$ 其中 $a \in \text{Act}_0$ 的情形类似可以得证.
- 对 $P = P_1 \circ P_2$ ($\circ \in \{+, ;\}$)，有 $s(P_1 \circ P_2) = s(P_1) \circ s(P_2)$，因此

  $$\begin{aligned} f(s(P_1 \circ P_2)) &= f(s(P_1)) \circ f(s(P_2)) \\ &\approx_{\text{eq}} s(P_1[\bar{a} \to P_a]) \circ s(P_2[\bar{a} \to P_a]) \\ &= s(P_1 \circ P_2[\bar{a} \to P_a]). \end{aligned}$$

- 对 $P = \mu x.P_1$，其操作语义模型 $s(\mu x.P_1)$ 中有可能包含回路，但仍然由基本的前缀操作和选择操作构成，因此可以转化成上述几种情况分别得到证明.

这样，就证明了 $f(s(P)) \approx_{\text{eq}} s(P[\bar{a} \to P_a])$.

定理 5.5.14 中的细化函数 f 称为安全的动作细化函数. 定理的结论说明，安全的语法和语义动作细化在弱交织迹等价和弱交织双模拟等价关系下关于结构化操作语义是一致的. 定理只在弱等价关系下成立是因为在语义细化中引入了内部动作转移的缘故.

例 5.5.15　设 $P = a.((1).b.0 + c.0), P_a = a_1.0 + (2).a_2.0$，在 P 中，有 $a \notin C(P)$，因此 $P[\bar{a} \to P_a]$ 是一个安全的细化操作. 令 $\text{Obs}\backslash\text{Act}_0 = \{a\}, f : \text{Obs}\backslash\text{Act}_0$ 是一个细化函数且 $f(a) = s(P_a)$，则 $s(P[\bar{a} \to P_a])$ 与 $f(s(P))$ 分别是安全的语法和语

义细化，它们的图形如图5.15所示. 从图中很容易验证，$f(s(P)) \approx_{\text{eq}} s(P[\bar{a} \to P_a])$，其中 $\text{eq} \in \{\text{it}, \text{ib}\}$.

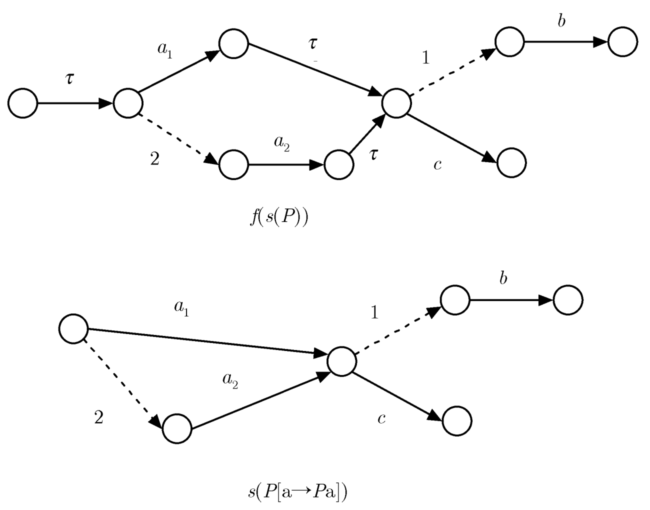


图 5.15 语法语义细化一致性的例子

语法语义细化的一致性定理有两方面的应用，一方面，语义细化后的模型可以由语法细化来刻画，另一方面，语法细化所刻画的系统可以由语义细化模型来实现，关于这两方面应用的详细讨论可参见文献 [71, 72]. 这样，就可以得到图 5.16 所示的一个安全的语法语义动作细化的转换图. 由此可见，我们定义的语法和语义细化具有良好的行为.

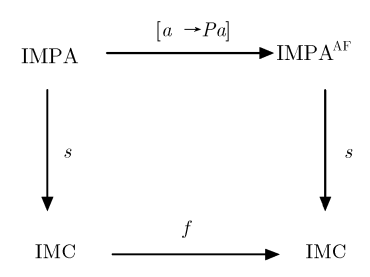


图 5.16 安全的语法语义动作细化相互转换图

第六章　模型检验

6.1　概　　述

本章将研究 IMC 模型作为一个功能与性能混合的并发模型在系统功能验证与性能评价方面的应用. 并发系统的功能分析与验证是传统并发理论研究的主要内容，它主要关心系统行为方面的特征，对系统的行为进行验证，而性能评价则主要研究并发系统数量上的性质，以各种量化的指标来对系统的动态行为作出评价，如系统的可靠性、稳定性等. 基本上，性能评价主要有两种方法，一种是基于测量的（measurement-based），另一种是基于模型的（model-based）. 在基于测量的方法中，通常是通过在一个具体的原型系统上进行模拟试验得出测试数据来进行分析的，而基于模型的方法则是构造一个能以足够精度描述感兴趣的度量的系统抽象模型，并在此模型上用分析、数值或者模拟的方法来分析系统数量上的性质.

模型检验是一种基于模型的形式验证方法. 它最早是由 Clarke 等人 [31,41] 及 Queille 等人 [90] 分别独立发展起来的，主要用于验证有限状态的系统是否满足一些特定的性质. 在模型检验中，系统模型通常用一个有限状态机的形式来表达，而与系统相关的性质通常用某种时序逻辑来表达，根据不同时序逻辑的不同表达能力，模型检验可以具有不同的功能. 最初的研究主要集中在对系统的功能行为进行验证，即系统质量方面的特征，如最终产生的结果是否正确，系统是否会导致死锁等等. 用于表达系统质的特征的时序逻辑有 LTL,CTL,CTL* 等，关于这些逻辑的模型检验算法被大量进行了研究并取得了丰富的成果. 对经典模型检验研究的综述可以参考文献 [33]. 这些研究成果的取得使得模型检验成为形式验证技术中最有吸引力的一种方法.

随着研究的不断深入和实际的需求，人们开始关注系统性能方面的性质，即数量上的特征，因此提出了各种带有数量指标的时序逻辑，如 RTCTL[42],PCTL[51], CSL[6,15,12] 等，利用这些逻辑，可以对很多系统性能方面的特征进行精确地表达和刻画，例如，用 CSL 可以表达类似这样的性质：系统在 24 小时内崩溃的概率小于 0.01；系统稳定运行在正常状态的概率大于 0.99 等. 这些反映系统性能特征的逻辑通常是在系统的性能模型上进行验证，其中 DTMC 和 CTMC 是两种最主要的性能模型，基于它们的模型检验算法也已经得到了很多相关的研究 [1,6,8~10,12,19,20,88,106]. 由于这些具有性能属性的时序逻辑强大的表达能力，使得模型检验成为了一种代表当前先进水平的（state-of-the-art）性能评价技术 [11,13,56].

然而，在以上介绍的取得很大成功的模型检验技术当中，所有的系统模型都是基于状态的（state-based），即它们都以系统所处的状态具有的性质为检验的基础.

相应地，以上所有的刻画逻辑也都是基于状态的. 随着系统复杂度和规模的增加，基于状态的系统很容易面临所谓的状态空间爆炸（state space explosion）问题，并且由于状态级的描述是一个平面模型，没有层次概念，当系统规模变得越来越大时，直接给出系统的状态模型是很困难的. 因此，这种基于状态的系统模型是不适合于分析大规模复杂系统的.

为了满足对现代大规模复杂系统的性能评价需求，需要有更加适合复杂系统性能分析的模型. 如本书前面所述，IMC 就是这样一个模型，它是完全基于动作，或者说是面向行为的（behaviour-oriented），因此可以以一种组合化的方式来对系统进行建模，大大降低系统的复杂性. 本章就研究如何将模型检验技术应用到 IMC 模型上. 由于上述的时序逻辑都是基于状态的，因此不能直接用于描述 IMC 上的性质，我们采用的是本书提出的 aCSL 逻辑来对 IMC 模型的时序性质做刻画. 我们在前面工作的基础上提出了 IMC 上的模型检验算法，它是现有的 CTMC 上的模型检验算法的一个推广，并且与现有算法相兼容，当 IMC 退化为 CTMC 时，我们的算法与现有的算法是一致的. 然而与现有算法相比，我们的算法是以系统执行的动作（或行为）为推理基础的，因此更适合组合化分析和验证大规模复杂系统.

6.2 基 本 原 理

本节首先简单介绍一下模型检验的基本原理和几种主要的方法，其中重点介绍基于语义的模型检验方法. 对这个过程的总体了解将有助于更好地理解本章的主要内容.

模型检验的基本原理可以用图 6.1 来直观地说明. 首先，要将待验证的系统抽象成一个有限状态机模型，同时将要验证的系统的性质用某种时序逻辑公式表达，然后通过一个有效的算法来检验系统模型是否满足该逻辑公式表达的性质，如果不满足，模型检验过程通常还可以给出一个反例来说明这个性质不成立的理由. 其中，实现模型检验算法的软件工具通常称为模型检验器（model checker）. 这个检验的过程是机械化自动的，不需要人工干预，具有较高的效率.

下面对图 6.1 涉及的几个主要的阶段作一简单说明.

- **建模阶段** 模型检验器的输入是两个形式模型，一个是表示系统的有限状态机模型，另一个是表示系统性质的时序逻辑公式. 因此，首先需要对实际系统进行建模，将其行为表示成某种有限状态机的形式. 而对于需要验证的系统的性质，也需要有一种精确的、没有歧义的方式来表示出来，这通常是通过某种时序逻辑来刻画的，这一过程实际上就是对系统性质描述的一个形式化的过程.

- **运行阶段** 模型检验器根据特定的算法，在输入的系统模型上进行检验，看所要验证的性质公式是否满足. 通常这种检验是一种穷尽搜索的方式，因此运行完毕可以保证检查的 100% 覆盖率. 根据具体实现算法的不同，不同模型检验器的效率也不一样.

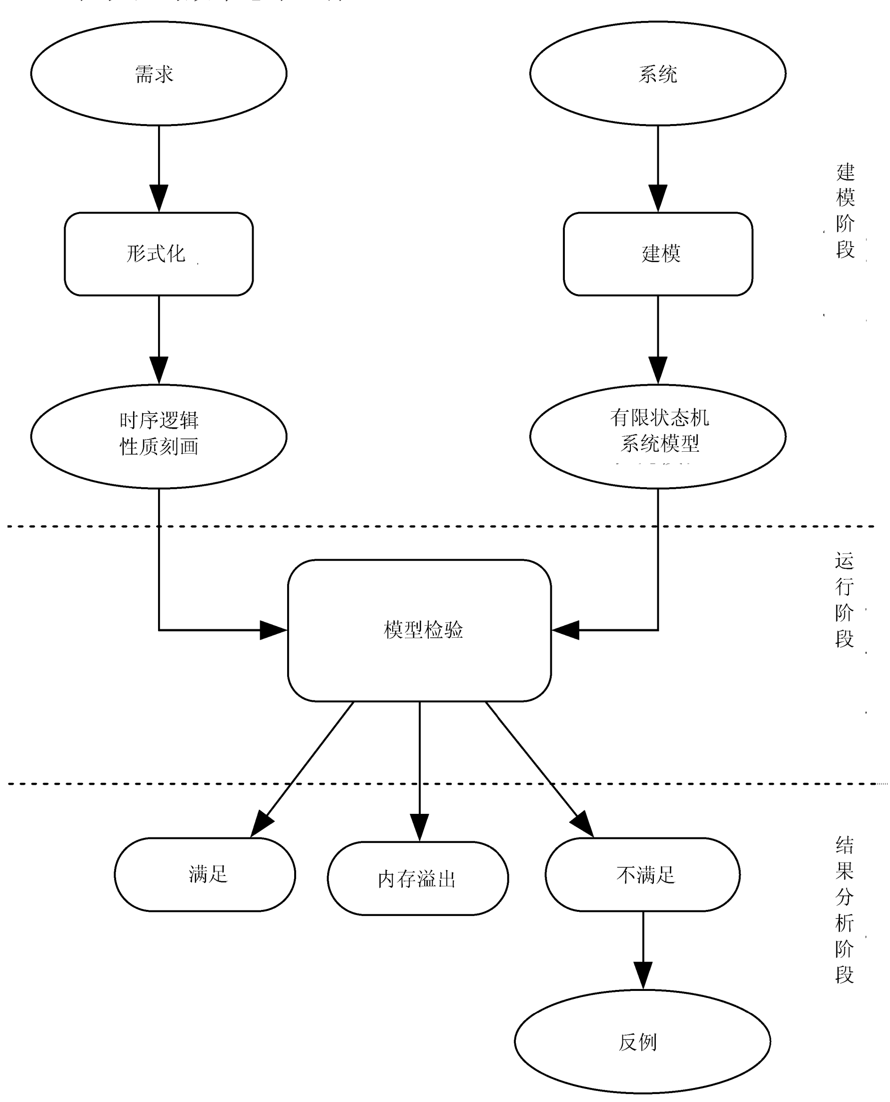


图 6.1 模型检验基本原理图

- **结果分析阶段** 这一阶段是对模型检测器的运行结果作分析，给出相应的结论. 基本上，可能的结果有三种情况：

(1) 系统满足给定的性质公式，则继续检验下一个性质（如果有的话）.

(2) 系统不满足给定的性质公式, 则根据情况给出相应的反例, 帮助定位错误.

(3) 内存不足, 无法完成检验. 这通常是由于系统状态空间过大, 超出机器的可用内存空间了, 因此需要减小需要验证的系统规模重新检验.

通过以上说明可以知道, 模型检验的核心是模型检验算法, 这个算法要解决的问题可以归纳为以下两大类问题 (即模型检验的核心问题):

- **全局模型检验问题** 给定一个有限状态系统模型 M 和一个时序逻辑公式 Φ, 找出 M 中所有满足 Φ 的状态集合.
- **局部模型检验问题** 给定一个有限状态系统模型 M, 一个时序逻辑公式 Φ 以及 M 中的一个状态 s, 判定 s 是否满足 Φ.

在局部模型检验问题中, 如果给定 M 的初始状态 s_0 满足公式 Φ, 则称模型 M 满足公式 Φ, 或者 M 是公式 Φ 的一个模型, 记为 $M \models \Phi$, 而 M 中某个状态 s 满足公式 Φ 记为 $s \models_M \Phi$, 当 M 在上下文已知的时候, 通常可以省略下标 M. 很显然, 全局模型检验问题的解包含了局部模型检验问题的解, 反过来, 对每个状态解决局部模型检验问题就构成了全局模型检验问题的解. 因此这两个问题是紧密相连的. 但是对于不同的应用, 它们有各自的适用场合. 例如, 当感兴趣的性质只与某些特定的初始状态有关时, 局部模型检验就更加适用, 因为在这种情况下, 通常涉及到的状态空间只是整个系统状态空间的一个子集, 因此可以在一定程度上避免状态空间爆炸问题.

以上是模型检验的基本原理, 它的具体实现有不同的方法, 主要有以下三种: 语义方法 (semantic approach)、自动机理论方法 (automata-theoretic approach) 和表方法 (tableau approach) . 不同的方法有不同的适用场合. 我们提出的 IMC 的模型检验方法是一种语义方法, 因此下面重点介绍基于语义方法的模型检验. 其他两种方法的模型检验可以参阅文献 [33, 81].

语义方法的基本思想是对于需要验证的逻辑公式, 在给定的有限状态机上递归的计算它的语义集合. 具体来说, 语义方法的一个关键思想是将每个逻辑公式根据它的语义与满足该公式的状态的集合相对应起来, 然后从公式最里层开始计算满足该子公式的状态集, 直到最后整个公式计算完毕为止.

例如, 对于以下的 CTL 公式

$$\mathbf{AG}\ \mathbf{EF}\ \mathbf{X}p,$$

计算的步骤是先计算满足原子命题 p 的状态集合, 然后根据算子 $\mathbf{X}$ 的语义计算 $\mathbf{X}p$ 对应的状态集合, 接着计算 $\mathbf{EF}$ 对应的状态集, 最后计算 $\mathbf{AG}$ 对应的状态集, 这样就把满足整个公式的状态集合找了出来. 可见, 这是一个解决全局模型检验问题的方法. 由于 CTL 公式具有良好的不动点 (fixed-point) 特征 [32], CTL 模型检验就可以简化为对 CTL 公式迭代求解不动点的计算, 因此这种方法有时候又称为迭代方法. 这种方法对于 CTL 模型检验具有很好的算法效率, 也是模型检验技术得以成

功的一个重要原因.

6.3 IMC 逻辑刻画的表达能力回顾

模型检验主要涉及两个方面，一是系统的表示，二是性质的刻画. 对一个有限状态系统来说，通常对系统性质的刻画采用的都是某种时序逻辑语言，因此, 本节先回顾一下 IMC 上的刻画逻辑 aCSL，重点放在其对 IMC 的时序性质的表达能力上.

aCSL 由状态公式和路径公式构成，其状态公式由以下语法产生：

$$\Phi ::= \text{true} \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie p}(\varphi),$$

其中 $p \in [0,1], \bowtie \in \{\leqslant, <, \geqslant, >\}$，而路径公式则由以下语法产生：

$$\varphi ::= \Phi_A \mathcal{U}^{<t} \Phi \mid \Phi_A \mathcal{U}^{<t}{}_B \Phi,$$

其中 $t \in \mathbb{R}_{>0}, A, B \subseteq \text{Act}$. 由 $\wedge$ 和 $\neg$ 可以得到其他所有的布尔连接词，如 $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2), \Phi_1 \rightarrow \Phi_2 = \neg\Phi_1 \vee \Phi_2$ 等. 其他线性时间路径公式如 X, U, G, F 等的推导请参考第三章.

与 CSL 一样，aCSL 逻辑主要用于刻画系统的性能与可靠性方面的性质，不同的是，在 CSL 中，系统的性能特征是基于状态的，而在 aCSL 中，所刻画的性能特征都与系统执行的动作有关，因此它既表达了系统的功能行为，又能给出这些行为执行时的性能特征. 另外一个与 CSL 不同的是，在 aCSL 中，没有反映系统长期运行以后处于稳定状态的概率特征的刻画（在 CSL 中，这个稳态特征用 $\mathcal{S}$ 算子来描述)，这是因为 IMC 与 CTMC 不同，由于动作转移与马尔可夫转移之间是独立执行的，相互之间并没有约束关系（与 SPA 模型不同，在 SPA 中，动作转移与延迟时间绑定在同一个关系中，其动作的执行必须限制在与其相联系的时间延迟之后)，因此，在 IMC 中，如果存在动作转移，那么像 CTMC 那样的稳定状态是不可能达到的.

在 aCSL 中，系统的性能与可靠性特征主要由 $\mathcal{P}$ 算子表达，由于我们关心的是系统的时序特征，因此 $\mathcal{P}$ 是作用在 aCSL 路径公式上的，它给出了一类广泛的基于路径的性能度量的刻画，这些路径包含了传统的 CTL 路径公式的集合. 以下用例子对一些常见的性质的刻画给出说明：

- **下一步（next）**：系统在 1 个时间单位内下一步执行动作 reset 的概率大于 0.8：

$$\mathcal{P}_{>0.8}(X^{<1}_{\text{reset}}\text{true}) = \mathcal{P}_{>0.8}(\text{true }_{\varnothing}\mathcal{U}^{<1}{}_{\text{reset}}\text{true}).$$

- **将来**（**future**）：设 $\Phi_{\rm down}$ 表示系统崩溃状态，则系统将在 24 个时间单位内崩溃的概率小于 0.01：

$$\mathcal{P}_{<0.01}(F^{<24}\Phi_{\rm down}) = \mathcal{P}_{<0.01}(\text{true }_{\rm Act}\mathcal{U}^{<24}\Phi_{\rm down}).$$

- **总是**（**globe**）：无论处于什么状态，系统都能在 12 个时间单位内通过执行动作 reset 回到初始状态 Φ_0，同时之前只执行 UP 集合里的动作：

$$\mathcal{P}_{\geqslant 1}({}_{UP}G^{<12}_{\rm reset}\Phi_0) = \neg\mathcal{P}_{\geqslant 1}({}_{UP}F^{<12}_{\rm reset}\neg\Phi_0) = \neg\mathcal{P}_{\geqslant 1}(\text{true }_{\rm UP}\mathcal{U}^{<12}{}_{\rm reset}\neg\Phi_0).$$

可见，aCSL 提供了非常丰富的反映系统性能特征的刻画，它不但可以描述经典的 CTL 路径公式的含义 (取动作集合为全集 Act，同时定义相应的状态上的原子命题即可)，而且能对系统特定的执行动作提供描述的手段，同时还提供了重要的性能量化指标 (如时间要求、概率特征等)，更重要的是，这些性质的刻画完全是形式化的，具有严格的语法和语义解释，消除了非形式化描述的模糊性和歧义性，因此 aCSL 具有很强的表达能力，是描述 IMC 的时序性能特征的理想语言.

6.4 模型检验算法

本节将给出 IMC 上的模型检验算法. 由 6.2 的介绍，可以将 IMC 的模型检验问题叙述如下：

> “给定一个IMC模型 $M = (S, \text{Act}, \longrightarrow, \dashrightarrow, s_0)$ 和一个aCSL状态公式 Φ，判定 M 是否满足 Φ，或者说 M 是否是 Φ 的一个模型，即 $M \models \Phi$ 是否成立”.

这实际上是一个局部模型检验问题，我们采用全局模型检验问题的解法来解决它，即先找出所有满足公式 Φ 的状态集合，然后通过判断 M 的初始状态 s_0 是否在 Φ 对应的状态集合里来决定 M 是否满足 Φ.

令 $\text{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$，即所有满足状态公式 Φ 的状态集合. 对于 aCSL 模型检验问题，我们采用的是与 CTL 模型检验一样的思想，对于一个给定的状态公式 Φ，递归的计算满足 Φ 的子公式的状态集合，由这些集合构造满足公式 Φ 的状态集 $\text{Sat}(\Phi)$.

6.4.1 基本布尔运算的计算

在 aCSL 状态公式中，其基本的构成包括真假值 (true, false)、通常的布尔连接词（$\wedge, \vee, \neg$ 等）和一个概率算子 $\mathcal{P}$，由于模型检验依赖于状态公式的构成，因此需要对状态公式中的每个组成部分定义相应的语义解释集合 $\text{Sat}(\Phi)$，才能够对其进行

递归分解计算. 本节先来看基本布尔运算是如何解释的，即这些基本布尔运算与集合 Sat(Φ) 之间的对应关系.

由 Sat(Φ) 的定义，可以在公式 Φ 与状态集合之间建立起以下对应关系：

$$
\begin{aligned}
\mathrm{Sat}(\mathrm{true}) &= \{s \mid s \models \mathrm{true}\} \\
&= S, \\
\mathrm{Sat}(\mathrm{false}) &= \{s \mid s \models \mathrm{false}\} \\
&= \varnothing, \\
\mathrm{Sat}(\Phi_1 \wedge \Phi_2) &= \{s \mid s \models \Phi_1 \wedge \Phi_2\} \\
&= \{s \mid s \models \Phi_1\} \cap \{s \mid s \models \Phi_2\} \\
&= \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2), \\
\mathrm{Sat}(\Phi_1 \vee \Phi_2) &= \{s \mid s \models \Phi_1 \vee \Phi_2\} \\
&= \{s \mid s \models \Phi_1\} \cup \{s \mid s \models \Phi_2\} \\
&= \mathrm{Sat}(\Phi_1) \cup \mathrm{Sat}(\Phi_2), \\
\mathrm{Sat}(\neg\Phi) &= \{s \mid s \models \neg\Phi\} \\
&= S - \{s \mid s \models \Phi\} \\
&= S - \mathrm{Sat}(\Phi).
\end{aligned}
$$

即 true 对应全集 S，false 对应空集 $\varnothing$，而通常的布尔运算 $\wedge, \vee, \neg$ 则分别对应集合的交、并、补运算. 这样，一个 aCSL 状态公式中所有的布尔运算都可以转换成状态集合上的基本操作（并、交、补等）.

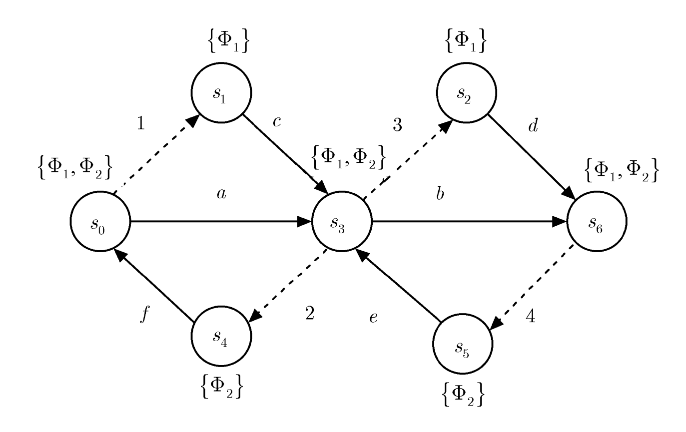


图 6.2 Sat(Φ) 的基本运算图例

例 6.4.1 如图6.2所示的一个IMC表示的系统，假设满足公式 Φ_1 与 Φ_2 的

状态集合已经计算完毕，分别标在相应的状态旁，即有

$$\mathrm{Sat}(\Phi_1) = \{s_0, s_1, s_2, s_3, s_6\},$$

$$\mathrm{Sat}(\Phi_2) = \{s_0, s_3, s_4, s_5, s_6\}.$$

则公式 $\Phi_1 \wedge \Phi_2, \Phi_1 \vee \Phi_2, \neg\Phi_1$ 所对应的状态集合分别为

$$\mathrm{Sat}(\Phi_1 \wedge \Phi_2) = \{s_0, s_3, s_6\},$$

$$\mathrm{Sat}(\Phi_1 \vee \Phi_2) = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\} = S,$$

$$\mathrm{Sat}(\neg\Phi_1) = \{s_4, s_5\}.$$

6.4.2 概率算子 $\mathcal{P}$ 的计算

在 IMC 的模型检验中，aCSL 状态公式中布尔运算操作的计算是与 CTL 一样简单而直接的，剩下的概率算子 $\mathcal{P}$ 的计算才是 IMC 模型检验的重点和关键. 本节将给出算子 $\mathcal{P}$ 的计算方法.

为了说明方便，先引入以下记号. 令 $s \not\longrightarrow$ 表示不存在从 s 出发的动作转移，再令

$$\mathbf{PS} = \{s \in S \mid s \not\longrightarrow\}, \quad \mathbf{NS} = S - \mathbf{PS}$$

分别表示概率状态（probabilistic state）集合与非确定状态（nondeterministic state）集合，即在 $\mathbf{PS}$ 中的状态只能执行马尔可夫转移，而在 $\mathbf{NS}$ 中的状态既可以执行马尔可夫转移，也可以执行动作转移. 对于 $\mathbf{NS}$ 中的状态 s，它将执行马尔可夫转移还是动作转移由系统在 s 的停留时间（由马尔可夫转移决定）和从 s 出发的动作转移的发生时间决定. 用 $\delta(s)$ 表示在状态 s 的停留时间，用 $\delta_A(s)$ 表示从 s 出发的 A 动作转移的发生时间，即

$$\delta_A(s) = \min(\{t \mid \exists s'(s \xrightarrow{(a,t)} s'), a \in A, A \subseteq \mathrm{Act}\}).$$

如果 $s \in \mathbf{PS}$，则对任意 $A \subseteq \mathrm{Act}$，定义 $\delta_A(s) = +\infty$. 另外，用 $\mathbf{I}(s, s')$ 来表示从 s 到 s 的动作转移所执行的动作，即 $(s, \mathbf{I}(s, s'), s') \in \longrightarrow$. 注意如果从 s 出发有多个可能的动作转移，则这些动作转移之间的选择是非确定的（因此该状态称为非确定状态）.

现在可以对概率算子 $\mathcal{P}_{\bowtie p}(\varphi)$ 的计算进行讨论了. 由 $\mathcal{P}_{\bowtie p}(\varphi)$ 的语义，它的计算依赖于以下对 $\mathrm{Prob}(s, \varphi)$ 的特征分析. 首先讨论 $\varphi = \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2$ 的情况. 分以下情形分别讨论：

情形 1 $s \models \Phi_2$. 由该路径公式的语义直接可得

$$\mathrm{Prob}(s, \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2) = 1.$$

情形 2 $(s \models \Phi_1 \wedge \neg\Phi_2) \wedge (s \in \mathbf{PS})$. 在这种情形下，$s$ 是个概率状态，即只能执行马尔可夫转移，因此，所有满足路径公式 φ 的路径都具有形式 $s \xrightarrow{(*,x)} \sigma'$，其中 $0 \leqslant x < t, \sigma' \in \mathrm{Path}(s')$. 因此有

$$\mathrm{Prob}(s, \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2) = \sum_{s' \in S} \int_0^t \boldsymbol{R}(s,s') \cdot e^{-E(s)\cdot x} \cdot \mathrm{Prob}(s', \Phi_1\ {}_A\mathcal{U}^{<t-x}\Phi_2)dx.$$

注意 $\boldsymbol{R}(s,s') \cdot e^{-E(s)\cdot x}$ 正是从 s 出发在时间 x 内转移到 s' 的概率密度.

情形 3 $(s \models \Phi_1 \wedge \neg\Phi_2) \wedge (s \in \mathbf{NS})$. 在这种情形下，$s$ 是非确定状态，则从 s 出发系统将来的行为有两种可能性，取决于在 s 上的停留时间和从 s 出发的 A 动作转移发生的时间. 如果 s 上的停留时间小于 A 动作转移发生的时间，即 $\delta(s) < \delta_A(s)$，则系统将执行马尔可夫转移，其行为与情形2相似：系统将在 $\delta_A(s)$ 时间内从 s 出发通过马尔可夫转移到达下一个状态. 否则，若 s 上的停留时间大于等于 A 动作转移发生的时间，即 $\delta(s) \geqslant \delta_A(s)$，则 A 动作转移将在 $\delta_A(s)$ 时刻发生，系统将以概率1通过一个 A 动作转移到达下一状态. 因此，有以下 $\mathrm{Prob}(s, \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2)$ 的计算：

$$\begin{aligned}\mathrm{Prob}(s, \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2) \quad = \quad & \sum_{\mathbf{R}(s,s') \geqslant 0} \int_0^{\delta_A(s)} \boldsymbol{R}(s,s') \cdot e^{-E(s)\cdot x} \cdot \mathrm{Prob}(s', \Phi_1\ {}_A \\ & \mathcal{U}^{<t-x}\Phi_2)\ dx + \sum_{\mathbf{I}(s,s') \in A} \mathrm{Prob}(s', \Phi_1\ {}_A\mathcal{U}^{<t-\delta_A(s)}\Phi_2).\end{aligned}$$

很显然，在其他所有的情况下，都有 $\mathrm{Prob}(s, \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2) = 0$.

综合以上情况，可以得到以下的结论：

定理 6.4.2 令 $F(s,t) = \mathrm{Prob}(s, \Phi_1\ {}_A\mathcal{U}^{<t}\Phi_2)$，则 $F(s,t)$ 满足以下方程：

$$F(s,t) = \begin{cases} 1, & \text{情形1} \\ \displaystyle\sum_{s' \in S} \int_0^t \boldsymbol{R}(s,s') \cdot e^{-E(s)\cdot x} \cdot F(s', t-x)dx, & \text{情形2} \\ \displaystyle\sum_{\boldsymbol{R}(s,s') \geqslant 0} \int_0^{\delta_A(s)} \boldsymbol{R}(s,s') \cdot e^{-E(s)\cdot x} \cdot F(s', t-x)dx & \\ \qquad + \displaystyle\sum_{\mathbf{I}(s,s') \in A} F(s', t-\delta_A(s)), & \text{情形3} \\ 0. & \text{其他情形} \end{cases}$$

对于路径公式 $\varphi = \Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$，情况稍微复杂一些，不过仍然与前面分析的相类似. 根据 $\Phi_1\ {}_A\mathcal{U}^{<t}{}_B\Phi_2$ 的语义，满足该路径公式的路径必须至少执行一步 B 动作转移从 Φ_1 状态到达 Φ_2 状态，因此满足 φ 的路径必须从 Φ_1 状态出发. 由此只需要讨论 $s \models \Phi_1$ 的情形.

情形 4 $(s \models \Phi_1) \wedge (\exists s'((s' \models \Phi_2) \wedge (\delta_B(s) \leqslant \delta_{A\backslash B}(s) < t)))$：由于 $\delta_B(s) \leqslant \delta_{A\backslash B}(s)$，因此，$B$ 动作转移可以发生而 $(A\backslash B)$ 动作转移不能发生（注意 A 与 B 可

能有交集)，又此时存在一个满足 Φ_2 的下一状态 s'，则根据语义，该状态必然满足该路径公式，因此有

$$\mathrm{Prob}(s, \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2) = 1.$$

情形 5 $(s \models \Phi_1) \wedge (s \in \mathbf{PS})$：与情形2类似，由于 s 是概率状态，只能执行马尔可夫转移，因此同理可得

$$\mathrm{Prob}(s, \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2) = \sum_{s' \in S} \int_0^t \boldsymbol{R}(s, s') \cdot e^{-E(s)\cdot x} \cdot \mathrm{Prob}(s', \Phi_1 \ {}_A\mathcal{U}^{<t-x}{}_B \Phi_2) dx.$$

情形 6 $(s \models \Phi_1) \wedge (s \in \mathbf{NS}) \wedge (\delta_{A\backslash B}(s) < \delta_B(s) < t)$：由于 $\delta_{A\backslash B}(s) < \delta_B(s)$，在这种情形下，$B$ 动作转移不能发生，因此系统只能执行 $A\backslash B$ 动作转移或者马尔可夫转移到达下一个仍然满足 Φ_1 的状态 s'，到底执行动作转移还是马尔可夫转移取决与系统在 s 上的停留时间 $\delta(s)$ 和 $A\backslash B$ 动作转移发生的时间 $\delta_{A\backslash B}(s)$. 因此，与情形3类似，有

$$\begin{aligned} & \mathrm{Prob}(s, \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2) \\ = \ & \sum_{\boldsymbol{R}(s,s') \geqslant 0} \int_0^{\delta_{A\backslash B}(s)} \boldsymbol{R}(s, s') \cdot e^{-E(s)\cdot x} \cdot \mathrm{Prob}(s', \Phi_1 \ {}_A\mathcal{U}^{<t-x}{}_B \Phi_2) \ dx \\ & + \sum_{\mathbf{I}(s,s') \in A\backslash B} \mathrm{Prob}(s', \Phi_1 \ {}_A\mathcal{U}^{<t-\delta_{A\backslash B}(s)}{}_B \Phi_2). \end{aligned}$$

显然，在其他所有情况下，$\mathrm{Prob}(s, \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2) = 0$.

综合以上情况，可以得到以下与定理 6.4.2 类似的结果.

定理 6.4.3 令 $G(s,t) = \mathrm{Prob}(s, \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B \Phi_2)$，则 $G(s,t)$ 满足以下方程：

$$G(s,t) = \begin{cases} 1, & \text{情形4} \\ \displaystyle\sum_{s' \in S} \int_0^t \boldsymbol{R}(s, s') \cdot e^{-E(s)\cdot x} \cdot G(s', t-x) dx, & \text{情形5} \\ \displaystyle\sum_{\boldsymbol{R}(s,s') \geqslant 0} \int_0^{\delta_{A\backslash B}(s)} \boldsymbol{R}(s, s') \cdot e^{-E(s)\cdot x} \cdot G(s', t-x) dx & \\ \qquad + \displaystyle\sum_{\mathbf{I}(s,s') \in A\backslash B} G(s', t - \delta_{A\backslash B}(s)), & \text{情形6} \\ 0. & \text{其他情形} \end{cases}$$

定理 6.4.2 和定理 6.4.3 给出了 $\mathrm{Prob}(s, \varphi)$ 在两种路径公式下的计算特征. 它们实际上都是递归方程，可以采用数值迭代的方法来进行求解. 本章稍后给出的算法实现就是采用的这种方法来计算 $F(s,t)$ 和 $G(s,t)$ 的.

另外，注意到在定理 6.4.2 和定理 6.4.3 中，若 $S = \mathbf{PS}$，即 IMC 中所有状态都是概率状态 (不存在动作转移)，此时，IMC 实际上退化成了一个 CTMC，因此定

理 6.4.2 和定理 6.4.3 中的情形 3 和情形 6 都不存在，剩下的情形实际上两个定理都是一致的，此时有 $F(s,t)=G(s,t)$ 且其满足的递归方程与已有的 CTMC 上的 CSL 模型检验的 $\mathcal{U}^{<t}$ 算子的计算是完全相同的 [15]，因此这里提出的 IMC 上的模型检验算法与现有的 CTMC 上的模型检验算法是完全兼容的.

另一个值得注意的情况是当 IMC 中不存在马尔可夫转移的时候，这时 IMC 退化成另一个极端 —— 功能模型 LTS，我们的算法仍然是有效的，而这时候 $F(s,t)$ 和 $G(s,t)$ 计算的是满足路径公式刻画的动作转移构成的路径的概率，其结果只有 0 和 1 两种情况，分别表示不满足和满足，即如果 $F(s,t)=0$，表示从 s 出发不存在满足要求的动作转移路径，$F(s,t)=1$ 则表示从 s 出发存在一条满足要求的动作转移路径.

上面两种极端情况表明 IMC 模型检验既具有性能评价的作用，也具有功能验证的作用，这得益于 IMC 模型自身的优点 —— 它正交结合了 CTMC 模型与 LTS 模型，因此能最大可能地保持两种模型自身的优点而不互相干扰.

表 6.1　$F(s,t)$ 的数值迭代算法

```
Algorithm ComputeF(S, R, A, Sat(Φ1), Sat(Φ2), t, k_max, ε, F)
 Input: S, R, A, Sat(Φ1), Sat(Φ2), t, k_max, ε
 Output: F

 begin
   forall s ∈ S, x < t
      F_0(s, x) := 0    //迭代初值
   endfor

   k := 0    //迭代计数器
   repeat    //开始迭代
   Δ_k := 0
     foreach s ∈ S    //计算 F_{k+1}
       switch(s)   //根据定理 6.4.2 的情形分情况计算
           case 1: F_{k+1}(s, t) := 1
           case 2: F_{k+1}(s, t) := INT(t, F_k)
           case 3: F_{k+1}(s, t) := INT(δ_A(s), F_k) + Σ_{I(s,s')∈A} F_k(s', t − δ_A(s))
           other cases: F_{k+1}(s, t) := 0
        endwitch
      Δ_{k+1} := Δ_k + (F_{k+1}(s, t) − F_k(s, t))    //计算总误差
     endfor
   k := k + 1
   until Δ_k ⩽ ε or k = k_max

   F := F_k
 end.
```

6.4.3 $\boldsymbol{F(s,t)}$ 与 $\boldsymbol{G(s,t)}$ 的计算

定理 6.4.2 和定理 6.4.3 所给出的递归积分方程直接求解是比较复杂的，但是利用计算机的数值计算优势，可以用迭代的方法给出 $F(s,t)$ 和 $G(s,t)$ 的近似数值解. 本节将给出一个基于这种方法的计算 $F(s,t)$ 与 $G(s,t)$ 的算法，并详细讨论它的具体实现.

表 6.1 和表 6.2 给出了计算 $F(s,t)$ 和 $G(s,t)$ 的算法的伪代码. 算法是直接基于两个定理进行计算的，其中每次迭代的核心是计算一个积分函数 $\textsc{Int}$. 算法中涉及到的函数 $\textsc{Int}(y,F)$ 表示以下形式的一个积分：

$$\int_0^y \sum_{\boldsymbol{R}(s,s')\geqslant 0} \boldsymbol{R}(s,s')\cdot e^{-E(s)\cdot x}\cdot F(s',t-x)dx.$$

表 6.2 $\boldsymbol{G(s,t)}$ 的数值迭代算法

```
Algorithm ComputeF(S, R, A, B, Sat(Φ1), Sat(Φ2), t,  k_max, ε, G)
 Input: S, R, A, B, Sat(Φ1), Sat(Φ2), t,  k_max, ε
 Output: G

 begin
   forall s ∈ S, x < t
      G_0(s, x) := 0    //迭代初值
   endfor

   k := 0    //迭代计数器
   repeat   //开始迭代
   Δ_k := 0
     foreach s ∈ S    //计算 G_{k+1}
       switch(s)   //根据定理 6.4.3 的情形分情况计算
          case 1: G_{k+1}(s, t) := 1
          case 2: G_{k+1}(s, t) :=INT(t, G_k)
          case 3: G_{k+1}(s, t) :=INT(δ_{A\B}(s), G_k) + Σ_{I(s,s')∈A\B} G_k(s', t − δ_{A\B}(s))
          other cases: G_{k+1}(s, t) := 0
       endswitch
     Δ_{k+1} := Δ_k + (G_{k+1}(s, t) − G_k(s, t))    //计算总误差
     endfor
   k := k + 1
   until Δ_k ⩽ ε or k = k_max

   G := G_k
 end.
```

这个积分可以用以下的近似求积公式来计算其近似的数值解：

$$\int_0^y f(x)dx \approx \sum_{j=0}^{N} \alpha_j\cdot f(x_j), \tag{6.1}$$

其中 $x_0, x_1, \cdots, x_N \in [0, y]$, $\alpha_0, \alpha_1, \cdots, \alpha_N$ 是不依赖于 f 的常数 (但有可能依赖于 N). 基于这种类型的求积公式的一个最简单的实现就是等距坐标法, 即取 $x_j = j \cdot h$, 其中 $h = t/N$, 称为步长. 这样, 就把积分区间等距地划分成了 N 等分. 很多著名的数值积分方法如梯形法、辛普森法、龙格 - 库塔法都属于这一类型的积分方法.

这一计算策略可以简化我们在表 6.1 和表 6.2 中提出的算法的实现, 根据这一思想, 可以用一个二维数组来代替二元函数 $F(s,t)$ 和 $G(s,t)$, 分别由下标 $s_0, \cdots, s_n$ 和 $x_0, \cdots, x_N$ 来索引. 在每次迭代过程中, 需要同时更新所有的 $F(s_i, x_i)$, 这样, 在迭代完成的时候, $F(s_i, x_N)$ 就包含了每个状态满足公式 φ 最终的概率 (即 $\text{Prob}(s_i, \varphi)$) .

基于上面所述的策略, 已经实现了表 6.1 和表 6.2 中算法的一个初步版本. 算法使用 C 语言实现, 对状态集合与动作集合采用的是显示表示方式. 具体的试验结果将在 6.5 的实例分析中给出.

6.4.4 IMC 模型检验算法

本小节将综合前面的讨论, 给出 IMC 上模型检验的完整算法流程. 如前所述, 我们采用的是基于语义方法的模型检验, 这是一种对要检验的公式递归分解计算的方法, 因此我们的算法以一个递归函数 Eval 为核心, 其输入是一个 aCSL 状态公式 Φ, 输出是满足该公式的状态集合 $\text{Sat}(\Phi)$.

在函数 $\text{Eval}(\Phi)$ 中需要对 Φ 的基本结构形式分别考虑. 根据状态公式的定义, 每一种逻辑操作算子都必须有相应的计算. 对于通常的布尔操作, 为了方便起见, 只列出了常见的 $\wedge, \vee, \neg, \rightarrow$ 运算的计算, 至于其他的逻辑运算, 可以转换成以上运算表示的等价公式再进行计算. 这些基本逻辑操作的计算根据 $\text{Sat}(\Phi)$ 的定义可以很直接地转换成集合上的基本操作.

对于状态公式中的概率算子 $\mathcal{P}$, 它的计算是 IMC 模型检验算法的核心. 根据路径公式的不同, 该算子的计算分为两种情况, 分别对应于 $F(s,t)$ 和 $G(s,t)$ 的计算, 计算的结果是每个状态 s_i 都对应一个相应的概率 $\text{Prob}(s_i, \varphi)$, 即从 s_i 出发满足 φ 的概率. 根据 $\mathcal{P}_{\bowtie p}(\varphi)$ 的语义, 此概率在 $\bowtie p$ 限定的范围内的状态即是满足该公式的状态, 因此对于 $\varphi = \Phi_1 \, {}_A\mathcal{U}^{<t} \Phi_2$, 有

$$\text{Sat}(\mathcal{P}_{\bowtie p}(\varphi)) = \{s \mid F(s,t) \bowtie p\},$$

同理, 对于 $\varphi = \Phi_1 \, {}_A\mathcal{U}^{<t}{}_B \Phi_2$, 有

$$\text{Sat}(\mathcal{P}_{\bowtie p}(\varphi)) = \{s \mid G(s,t) \bowtie p\}.$$

主算法则很简单, 首先由函数 $\text{Eval}(\Phi)$ 计算出满足 Φ 的状态集合, 然后通过判断初始状态 s_0 (或给定状态) 是否在 $\text{Sat}(\Phi)$ 中来回答模型检验问题. 由于 IMC 的模型检验主要关心的是系统的性能指标, 即 $\mathcal{P}$ 算子表达的性质, 对这些性能指标而言, 只要知道所关心的状态满不满足即可, 在模型检验作为性能评价手段时通常是不需要考虑反例的, 因此在我们的算法中并没有反例生成这一步.

最终的模型检验算法由表 6.3 给出. 算法的效率分析和优化问题将在 6.6 中详细讨论.

表 6.3 IMC 上的模型检验算法

```
Algorithm MC(M, Φ)   //M 是一个 IMC, Φ 是一个 aCSL 状态公式
 Input: M, Φ
 Output: Yes if M ⊨ Φ, else No

 function Eval(Φ)   //Eval 函数定义
 begin
   switch(Φ)   //根据公式 Φ 的形式分情况计算
      case true: return S
      case ¬Φ: return S − Eval (Φ)
      case Φ1 ∧ Φ2: return Eval (Φ1) ∩ Eval (Φ2)
      case Φ1 ∨ Φ2: return Eval (Φ1) ∪ Eval (Φ2)
      case Φ1 → Φ2 :return Eval (¬Φ1) ∪ Eval (Φ2)
      case P⋈p(φ):
        begin
          switch(φ)   //对两种路径公式分别处理
             case Φ1 AU<t Φ2: return {s | F(s, t) ⋈ p}
             case Φ1 AU<t B Φ2: return {s | G(s, t) ⋈ p}
        end
 end.   // Eval 函数定义结束

 begin   //主算法开始
    Sat(Φ) = Eval(Φ)
    if s0 ∈ Sat(Φ)
      return Yes
    else
      return No

 end.
```

6.5 实 例 分 析

本节将用两个实例来验证我们提出的算法. 第一个例子显示了如何将我们提出的模型检验算法应用到基于动作的系统上进行性能评价, 第二个例子是为了显示我们的算法与现有算法是一致的, 因此验证的是一个纯粹的 CTMC 上的性质, 试验结果表明我们的算法具有良好的兼容性, 它既能对一般的基于动作的系统进行性能评价, 同时对 IMC 的特例 ——CTMC 也具有同样的性能评价能力.

例 6.5.1 考虑一个由三个处理器 (processor) 和一个仲裁器 (voter) 构成的容错计算机系统. 其中, 三个处理器负责执行计算任务, 分别给出相应的计算结果, 然后由仲裁器根据多数原则在三个结果之中决定正确的值, 即选择相同最多的结果

为最终结果. 系统初始的时候，所有的组件的功能都是正常的. 但是三个处理器和仲裁器都有可能出现故障，假设一个单个的处理器故障率是平均每小时 λ 次，而仲裁器的故障率是平均每小时 ν 次. 如果出现故障，每个组件都可以在一定时间内修复，但是同一时间里只能有一个组件被修复，假设单个处理器的修复时间的期望值是 $1/\mu$ 小时，而仲裁器的修复时间期望值是 $1/\delta$ 小时. 系统只有在至少两个处理器和仲裁器都正常工作时才能正常工作，当仲裁器出现故障，或者三个处理器都出现故障的时候，系统将进入瘫痪状态，只能通过平均 $1/\delta$ 小时的时间让系统重置 (reset) 回到初始状态.

图6.3给出了这样一个容错计算机系统的IMC模型. 动作集合 $\text{Act} = \{F_1, F_2, F_3, F_v, \text{Rep}, \text{RESET}\}$, 其中 F_i 表示 i 个正常工作的处理器中的一个出现了故障 (failed), F_v 表示仲裁器出现了故障，Rep表示修复动作，RESET表示系统重置. 对于 $A \subseteq \text{Act}$，用 $\overline{A}$ 来表示 A 的 (关于 Act 的) 补集.

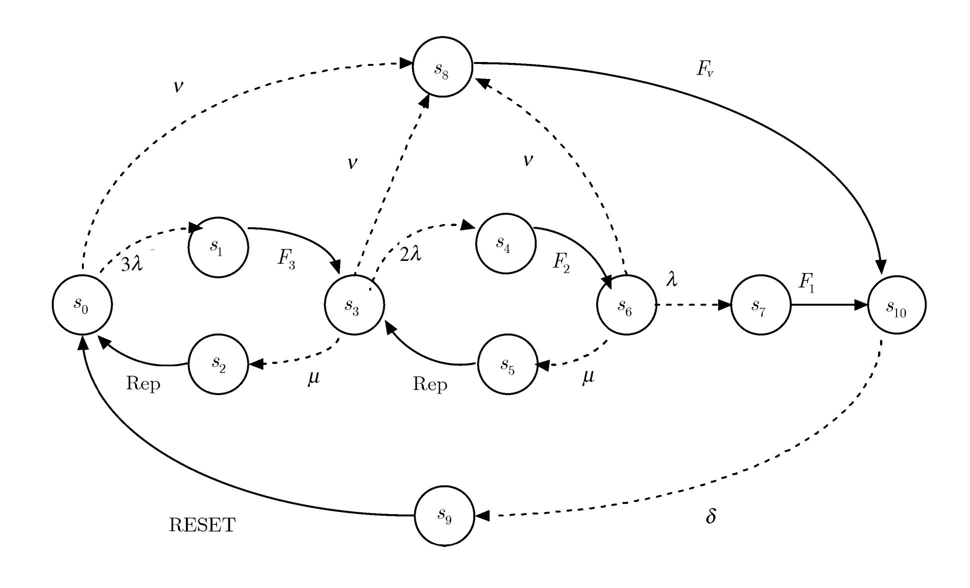


图6.3　一个容错计算机系统的IMC模型

现在，假设 $\lambda = 0.01, \mu = 0.02, \nu = 0.001, \delta = 0.2$，对 Act 中的每一个动作，它们的发生时间分别为 $0.3, 0.2, 0.1, 0.4, 0.1, 0.1$ (从到达它们可以发生的状态时算起). 用以下三条性质来说明如何用模型检验的方法给出系统的性能评价：

(1) 系统在10小时内瘫痪 (通过执行动作 F_1 和 F_v) 的概率小于0.02：

$$\Phi_1 = \mathcal{P}_{<0.02}(\text{ture}_{\overline{\{F_1,F_v\}}}\mathcal{U}^{<10}{}_{\{F_1,F_v\}}\text{ture});$$

(2) 系统在5小时内下一步只能执行 $\{F_3, F_2, \mathrm{Rep}\}$ 中的动作的概率大于0.1：

$$\Phi_2 = \mathcal{P}_{>0.1}(\mathrm{ture}_{\varnothing}\mathcal{U}^{<5}{}_{\{F_3,F_2,\mathrm{Rep}\}}\mathrm{ture});$$

(3) 系统在2小时内到达 Φ_2 状态的概率大于0.3：

$$\Phi_3 = \mathcal{P}_{>0.3}(\mathrm{ture}_{\mathrm{Act}}\mathcal{U}^{<2}\Phi_2).$$

第一条性质实际上是一个安全性性质（即系统在给定时间内在概率意义下不会发生瘫痪），而第二条性质描述了系统下一步（即 X 算子）动作的可能性，第三条则是对系统将来（即 F 算子）状态的一个估计.

我们在一台处理器为pentium4 1.7GHz、内存512MB、操作系统为Windows XP的 PC 上运行了该例子，结果显示在表6.4中. 对于性质 Φ_1 和 Φ_2，将近似求积公式(6.1) 的步长取为相同值，即 $h = 0.1$，试验表明这一取值对计算精度来说是足够的. 而对于性质 Φ_3，需要把步长 h 设为0.01才能得到更精确的结果，因为在性质 Φ_3 中计算积分的积分区间比较小，如果取 $h = 0.1$，则达不到计算精度要求. 如在表6.1和表6.2中的算法所描述的，在迭代计算中，用了一个误差 ε 来控制迭代，当两次迭代的结果误差小于 ε 时，认为迭代已经达到了一个不动点，此时的结果即是最后需要的结果. 在三个性质的迭代计算中，取的都是同一个误差控制值 $\varepsilon = 10^{-5}$. 由于状态空间比较小，只有11个状态数，因此所有性质的计算时间都在0.01秒内完成.

表 6.4 性质 Φ_1, Φ_2, Φ_3 的计算结果

Prob	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
Φ_1	0.011	0.018	0.011	0.018	0.093	0.018	0.093	1	1	0.011	0.007
Φ_2	0.134	1	1	0.172	1	1	0.089	0	0	0	0
Φ_3	1	1	1	1	1	1	0.042	0.289	0.275	1	0.314

最后，将表6.4中计算出来的Prob(s, φ) 的值与相应的概率界比较，就可以得到以下满足每条性质的状态集：

$$\mathrm{Sat}(\Phi_1) = \{s_0, s_1, s_2, s_3, s_5, s_9, s_{10}\},$$

$$\mathrm{Sat}(\Phi_2) = \{s_0, s_1, s_2, s_3, s_4, s_5\},$$

$$\mathrm{Sat}(\Phi_3) = \{s_0, s_1, s_2, s_3, s_4, s_5, s_9, s_{10}\}.$$

这样，就对性质 Φ_1, Φ_2 和 Φ_3 都给出了一个全局模型检验问题的解，即把每个状态满足的性质都计算了出来，通过检查给定的状态是否在所关心的性质对应的状态集里，就可以得到一个局部模型检验问题的解. 例如，由于 $s_0 \in \mathrm{Sat}(\Phi_1)$，可以得出结论 $s_0 \models \Phi_1$，它的含义就是如果系统从 s_0 状态开始运行，则它在10小时内发生瘫痪的概率小于0.02，换句话说，实际上系统从 s_0 出发开始运行在10小时内是不会趋向于瘫痪的，从而可以给出系统的一个可靠性保证.

通过以上例子可以看到，本章提出的 IMC 上的模型检验算法对基于动作的系统给出了一个非常直接与直观的验证方法. 它可以让我们在关注系统行为特征的同时得到对系统性能指标的评价，从而将 IMC 模型的性能评价功能最大程度的发挥了出来.

下面，再用一个例子说明我们提出的算法与现有的 CTMC 上的模型检验算法是兼容的，对于 IMC 的特例 ——CTMC，我们的模型检验算法仍然是适用的.

例 6.5.2 图6.4是改编自文献 [15] 的一个CTMC的例子，可以将其看成是IMC的一个特例，即 $\longrightarrow = \emptyset$，因此不存在动作以及动作转移. 其中，每个状态满足的性质标记在相应状态旁，即有

$$\mathrm{Sat}(\Phi_0) = \{s_0\}, \quad \mathrm{Sat}(\Phi_1) = \{s_1, s_3\}, \quad \mathrm{Sat}(\Phi_2) = \{s_2, s_3\}.$$

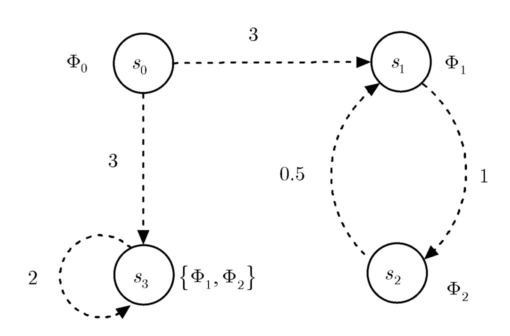


图6.4　一个CTMC表示的系统模型

现在假设我们关心这样一条性质：

$$\Phi = \mathcal{P}_{\geqslant 0.8}(\Phi_1 \mathcal{U}^{\leqslant 2} \Phi_2),$$

即要求系统在2个时间单位内最终到达 Φ_2 状态，并且之前只经过 Φ_1 状态的概率不小于0.8. 表6.5给出了在三种不同的积分步长（即 h）条件下的计算结果. 将表中结果与 Φ 中所要求的概率界0.8相比较, 可以得到满足 Φ 的状态集合为

$$\mathrm{Sat}(\Phi) = \{s_1, s_2\},$$

即有 $s_1 \models \Phi, s_2 \models \Phi$. 同时，注意到从 s_1 出发的满足 $\Phi_1 \mathcal{U}^{\leqslant 2} \Phi_2$ 的路径只有一条，即从 s_1 出发，在2个时间单位内到达 s_2 为止，该路径的概率根据定义可直接计算如下：

$$\int_0^2 e^{-1 \cdot x} dx = 1 - e^{-2} \approx 0.8647.$$

从运行结果来看，随着积分步长的缩小（从而积分计算更精确），$\mathrm{Prob}(s_1, \varphi)$ 的结果是越来越接近 $1 - e^{-2}$ 的.

上面的例子说明，我们提出的 IMC 上的模型检验算法对 CTMC 表示的模型同样适用，因此具有更广泛的通用性. 通过这样两个实例的运行，我们演示了如何用模型检验的方法对以 IMC 为模型的系统进行性能分析与评价. 通过 IMC 强大的建模能力，再结合 aCSL 丰富的性能性质描述能力，可以直接对基于动作的系统的性能特征进行分析和评价，由于基于动作的系统可以很方便的描述大规模复杂系统，因此我们提出的 IMC 上的模型检验方法具有潜在的更广泛的应用.

表 6.5 性质 Φ 的计算结果

$\text{Prob}(s_i,\varphi)$	s_0	s_1	s_2	s_3
h=0.1	0	0.8086	1	0
h=0.01	0	0.8603	1	0
h=0.001	0	0.8641	1	0

6.6 算法效率分析及优化考虑

本节对本章提出来的 IMC 上的模型检验算法的效率做一些分析，并对一些有用的优化技术作一定讨论.

6.6.1 算法效率分析

先分析算法的时间效率. 设系统状态空间的状态数为 $|S|$，状态公式的长度为 $|\Phi|$[1]. 由 6.3 中的算法可知，对于一个给定的 aCSL 状态公式 Φ，需要对其分解计算，每个子公式都要计算一次对应的满足集 Sat，因此总共需要计算 $|\Phi|$ 次. 对于通常的 $\wedge, \vee, \neg$ 等逻辑操作，很显然将其转化为状态集合上的操作需要花费 $O(|S|)$ 的时间. 对于概率算子 $\mathcal{P}$，其计算代价主要是 $F(s,t)$ 或 $G(s,t)$ 的计算，以 $F(s,t)$ 为例进行分析.

根据表6.1中的算法，对 $F(s,t)$ 的计算采用的是迭代的方法，其迭代次数由一个预先设置的迭代误差精度 ε 和最大迭代次数 $k_{\max}$ 共同决定，因此最坏情况下迭代的次数为 $k_{\max}$. 在每次迭代过程中，主要的计算就是 Int 函数，我们采用的是利用公式 (6.1) 进行近似积分的方法，根据公式 (6.1)，其计算代价由步长数 N 决定，即 $O(N)$. 因此 $F(s,t)$ 在最坏情况下的计算时间复杂度为 $O(k_{\max}\cdot N)$. 同理 $G(s,t)$ 的最坏情况计算复杂度也为 $O(k_{\max}\cdot N)$.

由此可知，本章提出的 IMC 上的模型检验算法在最坏情况下的时间复杂度为 $O(|\Phi|\cdot(|S|+k_{\max}\cdot N))$. 而在实际运行中，我们发现，若状态数不多，迭代次数通常都比较小，即算法能很快收敛到不动点.

[1]这里 Φ 的长度指的是 Φ 中子公式的数目，即公式中运算符的数目.

算法的空间复杂性方面，主要的存储开销有三方面，一是状态集合的存储，二是动作集合的存储，三是函数 $F(s,t)$ 及 $G(s,t)$ 的存储. 在我们的算法的实现中，状态集以及动作集采用的都是状态空间的显示存储技术，即状态数与存储单元是一一对应的，有多少状态数就需要多少存储单元来存储，因此所需要的存储空间分别为 $O(|S|)$ 和 $O(|\mathrm{Act}|)$. 对于 $F(s,t)$ 与 $G(s,t)$，由于把 $F(s,t)$ 和 $G(s,t)$ 都作为一个二维数组对待，其第一维坐标范围最大是整个状态空间 $|S|$，第二维坐标的最大范围是积分公式 (6.1) 中的步长数 N，因此所需的存储空间为 $O(|S|\cdot N)$. 由此可得算法总的空间复杂度为

$$O(|S|+|\mathrm{Act}|+|S|\cdot N)=O(|\mathrm{Act}|+|S|\cdot N).$$

可见，为了得到更精确的解，需要增大 N 的值 (即减小步长值 h)，从而就需要花费更大的存储代价.

表 6.6 给出了 IMC 上的模型检验的算法复杂度的一个总结.

表 6.6　IMC 模型检验的算法复杂度

算法	时间复杂度	空间复杂度
$F(s,t)$	$O(k_{\max}\cdot N)$	$O(\lvert\mathrm{Act}\rvert+\lvert S\rvert\cdot N)$
$G(s,t)$	$O(k_{\max}\cdot N)$	$O(\lvert\mathrm{Act}\rvert+\lvert S\rvert\cdot N)$
$MC(M,\Phi)$	$O(\lvert\Phi\rvert\cdot(\lvert S\rvert+k_{\max}\cdot N))$	$O(\lvert\mathrm{Act}\rvert+\lvert S\rvert\cdot N)$

6.6.2　优化考虑

从上面的分析可以看到，模型检验算法的时间和空间复杂度都与状态空间的大小 $|S|$ 有关. 由于模型检验的本质是对系统状态空间做穷尽搜索，因此对系统的规模比较敏感，在实际使用当中很容易碰到状态空间爆炸的问题. 所以要提高模型检验的实用性和效率，最大的挑战就是状态空间爆炸问题的解决（或者减轻）. 以下就具体讨论我们的算法在解决状态空间爆炸问题上可以采用的一些优化策略.

第一个可以采用的减轻状态空间爆炸问题的策略是等价关系的利用. 如在第四章看到的，IMC 上的互模拟等价关系可以将系统的状态空间按照等价关系划分，从而可以得到一个状态空间压缩的 IMC (如图 4.2 和 4.4 所示)，根据定理 4.4.1，在同一等价类中的状态具有相同的逻辑性质，即满足同样的 aCSL 公式，因此，模型检验问题可以转化到经过互模拟等价关系压缩的 IMC 上. 将这种方法以下面的定理形式严格叙述出来.

定理 6.6.1　令 $M=(S,\mathrm{Act},\longrightarrow,\dashrightarrow,s_0)$ 是一个IMC，R 是 M 上的一个强互模拟等价关系，$M/R=(S/R,\mathrm{Act},\longrightarrow_R,\dashrightarrow_R,[s_0]_R)$ 表示由 R 导出的压缩的IMC，其中 $[s]_R\xrightarrow{a}[s']_R$ 当且仅当 $s\xrightarrow{a}s'$，$[s]_R\overset{\boldsymbol{R}_R([s],[s'])}{\dashrightarrow}[s']_R$ 当且仅当 $\boldsymbol{R}_R([s],[s'])=\boldsymbol{R}(s,[s'])$，则

(1) 对于任何aCSL状态公式 Φ,

$$s \models_M \Phi \;\Leftrightarrow\; [s]_R \models_{M/R} \Phi,$$

(2) 对于任何aCSL路径公式 φ,

$$\mathrm{Prob}^M(s, \varphi) = \mathrm{Prob}^{M/R}([s]_R, \varphi).$$

根据上述定理, 在执行模型检验算法之前, 可以先找出系统状态空间上的一个互模拟等价关系 R, 然后根据 R 对系统进行状态空间的约简, 最后在约简以后的模型上进行模型检验, 上述定理即保证了在约简以后的模型上进行模型检验的结果跟原模型上进行检验的结果是一致的, 这样就在一定程度上减轻了状态空间爆炸的问题.

另一种比较常见同时也是使得模型检验得以在实际当中应用, 解决具有实际意义问题的方法是采用所谓的符号模型检验 (symbolic model checking) 的方法 [77], 这种方法通常是采用一种称为二叉判定图 (binary decision diagram, BDD) [27] 或者它的一些变体的结构来对状态空间进行编码存储, 因此这是一种基于状态空间的隐式表示的技术. 通过编码表示的状态空间与显式表示的状态空间相比需要的存储空间大大缩小, 使得模型检验能够处理的系统规模大大增加, 因此符号模型检验技术是一个减轻状态空间爆炸问题的非常重要和有效的手段.

在大规模的马尔可夫链的分析技术当中, 人们已经开始利用这种符号编码的技术来对马尔可夫链进行编码分析. 我们已经知道, 一个马尔可夫链由它的一步概率转移矩阵 (或转移率矩阵) 来决定, 因此这种符号编码技术主要解决的就是如何更有效地表示这些矩阵. MTBDD (multi-terminal binary decision diagram) [58,91] 是 BDD 的一种变体, 它可以有效地表示矩阵及其上的运算, 因此是符号化分析马尔可夫链的一个有力工具. 文献 [15] 又将其推广为 MTDD 并用于求解数值积分, 从而给出了 CTMC 上的一个近似符号模型检验算法. 我们相信, 结合马尔可夫链的 MTBDD 编码技术、状态空间的 BDD 编码技术和数值积分的 MTDD 计算方式, IMC 上的模型检验算法也可以符号化, 使得可以解决具有现实意义的系统评价和验证问题.

其他一些解决状态空间爆炸问题的技术包括对称约简、偏序约简等, 每种方法都有一个特定的适用场合, 这里就不再一一叙述了. 在我们第一版模型检验算法实现当中, 为了简单起见, 并没有过多的考虑这些优化的方法, 但是这些方法都是具有实际可操作性的, 因此我们将在以后的版本中考虑这些优化技术的使用, 以增强 IMC 模型检验的实用性.

参 考 文 献

[1] Aziz A, Singhal V, Brayton R K, Sangiovanni-Vincentelli A L. It usually works: The temporal logic of stochastic systems[C]. 7th International Conference On Computer Aided Verification,Lecture Notes in Computer Science. 1995,939:155–165.

[2] Aceto L. Action Refinement in Process Algebras[M]. University of Sussex: PhD thesis,1990.

[3] Aceto L, Hennessy M. Towards action-refinement in process algebras[C]. Proceedings of the Fourth Annual Symposium on Logic in computer science.IEEE Press,1989,138–145.

[4] Aceto L, Hennessy M. Adding action refinement to a finite process algebra[J]. Inf. Comput.,1994, 115(2): 179 – 247.

[5] Andova S, Baeten J C M. Abstraction in probabilistic process algebra[C]. TACAS'01, Lecture Notes in Computer Science, 2001,2031: 204–219.

[6] Aziz A, Sanwal K, Singhal V, Brayton R K. Model-checking continuoustime Markov chains[J]. ACM Transactions on Computational Logic, 2000,1(1): 162–170.

[7] Baeten J C M, Bergstra J A. Real time process algebra[J]. Formal Aspects of Computing,1991, 3(2): 142–188.

[8] Baier C, Clarke E. Hartonas-Garmhausen, Kwiatkowska M, Ryan. Symbolic model checking for probabilistic processes[C]. Annual International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science,1997, 1256: 430–440.

[9] Baier C, Haverkort B R, Hermanns H, Katoen J P. Model checking continuous-time Markov chains by transient analysis[C]. CAV'00, Lecture Notes in Computer Science, 2000, 1855: 358–372.

[10] Baier C, Haverkort B R, Hermanns H, Katoen J P. On the logical characterisation of performability properties[C]. ICALP'00, Lceture Notes in Computer Science,2000,1853: 780–792.

[11] Baier C, Haverkort B R, Hermanns H, Katoen J P. Automated Performance and Dependability Evaluation Using Model Checking[M]. Mariacarla Calzarossa and Salvatore Tucci, editors. Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures. Lecture Notes in Computer Science,2002, 2459: 261–289.

[12] Baier C, Haverkort B R, Hermanns H, Katoen J P. Model-checking algorithms for continuous-time Markov chains[J]. IEEE Trans. Software Eng., 2003, 29(6): 524–541.

[13] Baier C, Haverkort B R, Hermanns H, Katoen J P. Model checking meets performance evaluation[J]. SIGMETRICS Performance Evaluation Review, 2005, 32(4): 10–15.

[14] Baier C, Hermanns H. Weak bisimulation for fully probabilistic processes[C]. Computer Aided Verification, 9th International Conference, Lecture Notes in Computer Science, 1997, 1254: 119–130.

[15] Baier C, Katoen J P, Hermanns H. Approximate symbolic model checking of continuous-time Markov chains[C]. Concurrency Theory, 10th International Conference, Lecture Notes in Computer Science, 1999,1664: 146–161.

[16] Baier C, Katoen J P, Hermanns H, Haverkort B. Simulation for continuous-time Markov chains[C]. Concurrency Theory,13th International Conference, Lecture Notes in Computer Science,2002,2421: 338–354.

[17] Baier C, Katoen J P, Hermanns H, Haverkort B R. Simulation for Continuous-time Markov Chains[M]. Uni.Bonn:Technical Report, 2002.

[18] Baier C, Katoen J P, Hermanns H, Wolf V. Comparative branching-time semantics for Markov chains[J]. Inf. Comput., 2005,200(2): 149–214.

[19] Baier C, Kwiatkoswka M. Model checking for a probabilistic branching time logic with fairness[J]. Distributed Computing, 1998,11(3): 125–155.

[20] Beauquier D, Slissenko A. Polytime model checking for timed probabilistic computation tree logic[J]. Acta Informatica, 1998, 35: 645–664.

[21] Bernardo M, Gorrieri R. Extended Markovian process algebra[C]. Concurrency Theory, 7th International Conference, Lecture Notes in Computer Science, 1996,1119: 315–330.

[22] Bernardo M, Gorrieri R. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time[J]. Theoretical Computer Science, 1998,202: 1–54.

[23] Bolognesi T, Brinksma E. Introduction to the ISO specification language LOTOS[J]. Computer Networks and ISDN Systems, 1987, 14(1): 25–59.

[24] Bowman H, Derrick J. Extending LOTOS with time: True concurrency perspective[C]. In AMAST Workshop on Real-Time Systems, Lecture Notes in Computer Science, 1997,1231, 382–399.

[25] Bravetti M. Revisiting interactive Markov chains[J]. Electronic Notes in Theoretical Computer Science,2002, 68(5).

[26] Browne M C, Clarke E M, Grümberg O. Characterizing finite kripke structures in propositional temporal logic[J]. Theoretical Computer Science,1988, 59: 115–131.

[27] Bryant R E. Graph-based algorithms for Boolean function manipulation[J]. IEEE Transactions on Computers, 1986,C-35: 677–691.

[28] Hoare C A R. Communicating Sequential Processes[M]. Upper Saddle River, NJ, USA: Prentice Hall,1985.

[29] Stirling C. Modal and temporal logics[M]. S. Abramsky. Dov. M. Gabbay, and T. S. E. Maibaum, editors. Handbook of Logic in Computer Science,Oxford: Clarendon Press, 1992, 2(Background: Computational Structures): 477–563.

[30] Hoare C A R. Communicating sequential processes[J]. Communications of the ACM, 1978, 21(8): 666–677.

[31] Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching-time temporal logic[C]. Proc. Logic of Programs, Lecture Notes in Computer Science, 1981,131: 52–71.

[32] Clarke E M, Emerson E A, Prasad Sistla A. Automatic verification of finite-state concurrent systems using temporal logic specifications[J]. ACM Trans. Program. Lang. Syst., 1986, 8(2): 244–263.

[33] Clarke E M, Grumberg O, Peled D A. Model Checking[M]. Cambridge, Massachusetts: The MIT Press,1999.

[34] Conway A E, Georganas N D. Queueing Networks – Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation[M]. Cambridge, Massachusetts: The MIT Press,1989.

[35] Czaja I, van Glabbeek R J, Goltz U. Interleaving semantics and action refinement with atomic choice[C]. Proc. Advances in Petri Nets: The DEMON Project, Lecture Notes in Computer Science, 1992, 609: 89-107.

[36] de Bakker J W, Erik P. de Vink. Bisimulation semantics for concurrency with atomicity and action refinement[J]. Fundam. Inform, 1994, 20(1/2/3): 3–34.

[37] Degano P, Gorrieri R. A causal operational semantics of action refinement[J]. Information and Computation, 1995, 122(1): 97–119.

[38] Desharnais J, Gupta V, Jagadeesan R, Panangaden P. Weak bisimulation is sound and complete for PCTL* [C]. // Concurrency Theory, 13th International Conference, Lecture Notes in Computer Science,2002, 2421: 355–370.

[39] Desharnais J, Panangaden P. Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes[J]. J. Log. Algebr. Program, 2003, 56(1-2): 99–115.

[40] Emerson E A. Temporal and Modal Logic[M]. Jan van Leeuwen, editor. Handbook of Theoretical Computer Science. Amsterdam: North-Holland,1990,Volume B: 995–1072.

[41] Emerson E A, Clarke E M. Using branching time temporal logic to synthesize synchronization skeletons[J]. Science in Computer Programming,1982, 2(3): 241–266.

[42] Emerson E A, Mok A K, Sistla A P, Srinivasan J. Quantitative temporal reasoning[J]. Real-Time Syst., 1992, 4(4): 331–352.

[43] Fecher H. A real-time process algebra with open intervals and maximal progress[J]. Nord. J. Comput,2001. 8(3): 346–365.

[44] Fecher H, Majster-Cederbaum M E, Wu J. Action refinement for probabilistic processes with true concurrency models[C]. PAPM-PROBMIV'02, Lecture Notes in Computer Science, 2002, 2399: 77–94.

[45] Fecher H, Majster-Cederbaum M E, Wu J. Refinement of actions in a real-time process algebra with a true concurrency model[J]. Electronic Notes in Theoretical Computer Science, 2002,70(3): 620–640.

[46] Goltz U, Gorrieri R, Rensink A. On syntactic and semantic action refinement[C]. TACS'94, Lecture Notes in Computer Science, 1994, 789: 385–404.

[47] Goltz U, Gorrieri R, Rensink A. Comparing syntactic and semantic action refinement[J]. Information and Computation, 1996,125: 118–143.

[48] Gorrieri R. Refinement, Atomicity and Transactions for Process Description Languages[M]. Dipartimento di Informatica, Università di Pisa: Dissertazione per il titolo di dottore di ricerca in informatica (terzo ciclo), 1990.

[49] Gorrieri R. A hierarchy of system descriptions via atomic linear refinement[J]. Fundam. Inform, 1992, 16(3-4): 289–336.

[50] Gorrieri R, Rensink A. Action Refinement[M]. University of Bologna, Department of Computer Science: Technical Report UBLCS-99-9,1999.

[51] Hansson H, Jonsson B. A logic for reasoning about time and reliability[J]. Formal Asp. Comput., 1994,6(5): 512–535.

[52] Hennesy M C B, Milner R. Algebraic laws for nondeterminism and concurrency[J]. Journal of the ACM, 1985,32(1): 137–161.

[53] Hermanns H. Interactive Markov Chains[M]. Universität Erlangen-Nürnberg: PhD thesis, 1998.

[54] Hermanns H, Herzog U, Katoen J P. Process algebra for performance evaluation[J]. Theoretical Computer Science,2002, 274(1-2): 43–87.

[55] Hermanns H, Herzog U, Mertsiotakis V. Stochastic process algebras - between LOTOS and Markov chains[J]. Computer Networks, 1998,30(9-10): 901–924.

[56] Hermanns H, Katoen J P. Performance evaluation: = (process algebra + model checking) × Markov chains[C]. Concurrency Theory, 12th International Conference, Lecture Notes in Computer Science, 2001,2154: 59–81.

[57] Hermanns H, Katoen J P, Meyer-Kayser J, Siegle M. Towards model checking stochastic process algebra[C]. Integrated Formal Methods, Second International Conference, Lecture Notes in Computer Science, 2000, 1945: 420–439.

[58] Hermanns H, Meyer-Kayser J, Siegle M. Multi-terminal Binary Decision Diagrams to Represent and Analyse Continuous-time Markov Chains[M].Univ. Twente: Technical Report, 1999.

[59] Hermanns H, Rettelbach M. Syntax, Semantics, Equivalences, and Axioms for MTIPP [M]. Uni. Erlangen-Nürnberg: Technical Report,1994.

[60] Herzog U. Formal description, time and performance analysis-A framework[J]. Entwurf und Betrieb verteilter Systeme, Informatik-Fachberichte, 1990,264: 172–190.

[61] Hillston J. PEPA: Performance Enhanced Process Algebra[M].University of Edinburgh, Edinburgh, Scotland: Technical Report CSR-24-93,1993.

[62] Hillston J. A Compositional Approach to Performance Modelling[M]. New York, NY, USA: Cambridge University Press, 1996.

[63] Huhn M. Action refinement and property inheritance in systems of sequential agents[C]. Concurrency Theory, 7th International Conference, Lecture Notes in Computer Science, 1996, 1119: 639–654.

[64] ISO.LOTOS - language of Temporal Ordering Specification[M]. ISO: Technical Report ISO DP 8807, 1987.

[65] Jonsson B, Larsen K G. Specification and refinement of probabilistic processes[C]. Proceedings 6th Annual Symposium on Logic in Computer Science, IEEE Press, 1991,266–277.

[66] Jou C C, Smolka S A. Equivalences, congruences, and complete axiomatizations for probabilistic processes[C]. CONCUR'90, Lecture Notes in Computer Science, 1990, 458: 367–383.
[67] Katoen J P. Quantitative and Qualitative Extensions of Event Structures[M]. Universität Twente:PhD thesis, 1996.
[68] Kozen D. Results on the propositional μ-calculus[J]. Theoretical Computer Science, 1983,27(1): 333–354.
[69] Larsen K G, Skou A. Bisimulation through probabilistic testing[J]. Information and Computation, 1991,94(1): 1–28.
[70] Majster-Cederbaum M, Wu J. Towards action refinement for true concurrent real time[J]. Acta Inf., 2003, 39(8): 531–577.
[71] Majster-Cederbaum M E, Salger F. Correctness by construction: Towards verification in hierarchical system development[C]. SPIN'00, Lecture Notes in Computer Science, 2000,1885: 163–180.
[72] Majster-Cederbaum M E, Salger F, Sorea M. A priori verification of reactive systems[C]. FORTE'00, IFIP Conference Proceedings, 2000, 183: 35–50.
[73] Majster-Cederbaum M E, Wu J. Action refinement for true concurrent real time[C]. Proc. ICECCS'01, IEEE Computer Society,2001, 58-68.
[74] Majster-Cederbaum M E, Wu J. Adding action refinement to stochastic true concurrency models[C]. ICFEM'03, Lecture Notes in Computer Science, 2003, 2885: 226–245.
[75] Majster-Cederbaum M E, Wu J, Yue H, Zhan N. Refinement of actions for real-time concurrent systems with causal ambiguity[C]. ICFEM'04, Lecture Notes in Computer Science, 2004, 3308: 449–463.
[76] Marsan M A, Balbo G, Conte G. et al. Modelling with Generalized Stochastic Petri Nets[M]. New York: Wiley, Wiley series in parallel computing, 1995.
[77] McMillan K L. Symbolic Model Checking: An Approach to The State Explosion Problem[M]. Dordrecht, Netherlands: Kluwer Academic Publishers, 1993.
[78] Meyer J F, Movaghar A, Sanders W H. Stochastic activity networks: Structure, behavior, and application[C]. PNPM'85, IEEE Computer Society,1985,106–115.
[79] Milner R. A Calculus of Communicating Systems[M]. New York: Springer-Verlag, 1982.
[80] Milner R. Communication and Concurrency[M]. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.
[81] Müller-Olm M, Schmidt D, Steffen B. Model-checking: A Tutorial Introduction[M]. Agostino Cortesi and Gilberto Filé, editors.Static analysis. Lecture Notes in Computer Science, 1999,1694:330–354.
[82] Nielsen M, Engberg U, Larsen K S. Fully abstract models for a process language with refinement[C]. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, London, UK, 1989, 523–548.
[83] Nounou N. A Methodology for Specification-based Performance Analysis of Protocols [M]. Texas A&M Uni.:PhD thesis, 1986.

[84] Nounou N, Yemini Y. Algebraic specification-based performance analysis of communication protocols[C]. Proc.PSTV'84, North-Holland, 1984,541–560.

[85] Philippou A, Lee I, Sokolsky O. Weak bisimulation for probabilistic systems[C]. 11th International Conference on Concurrency Theory, Lecture Notes in Computer Science, 2000,1877: 334-349.

[86] Plateau B, Atif K. Stochastic automata network for modeling parallel systems[J]. IEEE Trans. on Softw. Eng., Special Section on Parallel Systems Performance, 1991,17(10): 1093.

[87] Plotkin G. A Structural Approach to Operational Semantics[M]. Computer Science Department, Aarhus University: Technical Report DAIMI FN-19, 1981.

[88] Pnueli A, Zuck L D. Probabilistic verification[J]. Inf. Comput, 1993, 103(1): 1–29.

[89] Qin G, Wu J. Action refinement for real-time concurrent processes with urgency[J]. Electronic Notes in Theoretical Computet Science, 2005,139(1): 123–144.

[90] Queille J P, Sifakis J. Specification and verification of concurrent systems in CESAR[C]. Proceedings of the Fifth International Symposium in Programming, Lecture Notes in Computer Science, 1982, 137: 337–351.

[91] Bahar R I, Frohm E A, Gaona C M, Hachtel G D, Macii E, Pardo A, Somenzi F. Algebraic decision diagrams and their applications[C]. IEEE/ACM International Conference on CAD, IEEE Computer Society Press, 1993, 188–191.

[92] Säidouni D E, Courtiat J P. Syntactic action refinement in presence of multiway synchronization[C]. Proceedings of the International Workshop on Semantics of Specification Languages, Workshops in Computing, London, UK, 1994, 289–303.

[93] Segala R, Lynch N A. Probabilistic simulations for probabilistic processes[J]. Nord. J. Comput, 1995, 2(2): 250–273.

[94] Stewart W J. Introduction to The Numerical Solution of Markov Chains[M]. Princeton: Princeton University Press, 1994.

[95] van Glabbeek R, Goltz U. Partial orders semantics for refinement of actions – neither necessary nor always sufficient but appropriate when used with care[J]. Bulletin of the European Association for Theoretical Computer Science, 1989, 38: 154–163.

[96] van Glabbeek R, Goltz U. Well-behaved flow event structures for parallel composition and action refinement[J]. Theoretical Computer Science, 2004,311(1-3): 463–478.

[97] van Glabbeek R J. The linear time - branching time spectrum I[M]. Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. Handbook of Process Algebra, Chapter 1,Dordrecht, The Netherlands: Elsevier Science, 2001: 3–100.

[98] van Glabbeek R J. Comparative Concurrency Semantics and Refinement of Actions[M]. Free University of Amsterdam, Amsterdam: PhD thesis, 1990.

[99] van Glabbeek R J. The refinement theorem for ST-bisimulation semantics[C]. Proceedings IFIP TC2 Working Conference on Programming Concepts and Methods, Sea of Gallilea, Israel. North-Holland, 1990.

[100] van Glabbeek R J. The linear time - branching time spectrum II: The semantics of sequential systems with silent moves[C]. CONCUR'93, Lecture Notes in Computer Science, 1993,715: 66–81.

[101] van Glabbeek R J, Goltz U. Refinement of actions in causality based models[C]. Proc. REX Workshop, Lecture Notes in Computer Science, 1989, 430: 267–300.

[102] van Glabbeek R J, Goltz U. Equivalences and refinement[J]. Lecture Notes in Computer Science,1990,469: 309–333.

[103] van Glabbeek R J, Goltz U. Refinement of Actions in Causality Based Models[M]. J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors. Proceedings REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness, Lecture Notes in Computer Science, 1990,430: 267–300.

[104] van Glabbeek R J, Goltz U. Refinement of actions and equivalence notions for concurrent systems[J]. Acta Inf., 2001, 37(4/5): 229–327.

[105] van Glabbeek R J, Smolka, Steffen. Reactive, generative, and stratified models of probabilistic processes[J]. Information and Computation, 1995, 121: 59–80.

[106] Vardi M Y. Automatic verification of probabilistic concurrent finite-state programs[C]. Proc. FOCS'85, 1985, 327–338.

[107] Vogler W. Failures semantics based on interval semiwords is a congruence for refinement[J]. Distributed Computing, 1991, 4(3): 139–162.

[108] Vogler W. Action refinement and bisimulation[J]. J-LECT-NOTES-COMP-SCI, 1992, 625: 183–216.

[109] Winskel G. An introduction to event structures[C]. Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, REX'88, Lecture Notes in Computer Science,1988, 354: 364–379.